```
Hello, my name is Daniel Asa and I came here to teach you JavaScript //

JavaScript Output //

JavaScript Display Possibilities //
:JavaScript can "display" data in different ways //

.Writing into an HTML element, using innerHTML //
.()Writing into the HTML output using document.write //
.()Writing into an alert box, using window.alert //
()Writing into the browser console, using console.log //

Using innerHTML //

To access an HTML element, JavaScript can use the //
document.getElementById(id) method

To use innerHTML, you must first create a tag with the desired id, //
.then run the following code inside your script code

('document.getElementById('id').innerHTML = ('hi //

:()For testing purposes, it is convenient to use document.write //

Note: Using document.write() after an HTML document is loaded, will //
:delete all existing HTML

('document.write('Daniel //

:You can use an alert box to display data //

('♥ alert('Hi, How are you //

For debugging purposes, you can call the console.log() method in the //
.browser to display data

Note:console.log is run in your browser section, you must //
right-click and click inspect, and the console section

('console.log('hi //

.JavaScript does not have any print object or print methods //
```

```
.You cannot access output devices from JavaScript //

The only exception is that you can call the window.print() method in //
.the browser to print the content of the current window

()print //

JavaScript Statements //

:let, var, const //

'let a = 'hi //
'var b = 'hello //
'const c = 'he //

Note: The variable is converted from lat to var, but not from var to //
lat

; Semicolons //
.Semicolons separate JavaScript statements //

;a = 1 //
;b = 2 //
;c = 3 //

;a = 1; b = 2; c = 3 //

JavaScript Code Blocks //

JavaScript statements can be grouped together in code blocks, inside //
.{...} curly brackets

The purpose of code blocks is to define statements to be executed //
.together

One place you will find statements grouped together in blocks, is in //
:JavaScript functions

In JavaScript you cannot use these reserved words as variables, //
:labels, or function names

abstract, arguments, await*, boolean //
break, byte, case, catch //
```

```
char, class*,  const,  continue //
debugger, default, delete, do //
double,  else, enum*,  eval //
export*, extends*, false,  final //
finally, float,  for function //
*goto, if, implements, import //
in,  instanceof, int, interface //
let*, long,  native, new //
null, package, private,  protected //
public,  return, short,  static //
super*,  switch, synchronized, this //
throw, throws, transient,  true //
try, typeof, var, void //
volatile, while, with, yield //

.Words marked with* are new in ECMAScript 5 and 6 //

Removed Reserved Words //
The following reserved words have been removed from the ECMAScript //
:5/6 standard
abstract, boolean, byte, char //
double,  final,  float,  goto //
int, long, native, short //
synchronized, throws, transient, volatile //

JavaScript Syntax //

JavaScript Values //
:The JavaScript syntax defines two types of values //

Fixed values //
Variable values //
.Fixed values are called Literals //

.Variable values are called Variables //

JavaScript Literals //
:The two most important syntax rules for fixed values are //

Numbers are written with or without decimals .1 //

10.50 //
1001 //
```

```
:Strings are text, written within double or single quotes .2 //

"String" //
'string' //

JavaScript Variables //
.In a programming language, variables are used to store data values //

JavaScript uses the keywords var, let and const to declare //
.variables

.An equal sign is used to assign values to variables //

In this example, x is defined as a variable. Then, x is assigned //
:(given) the value 6

let x //
x = 6 //

JavaScript Operators //
:JavaScript uses arithmetic operators ( + - * / ) to compute values //

document.write(5+6) *10 /2 //

JavaScript uses an assignment operator ( = ) to assign values to //
:variables

let x, y //
x = 5 //
y = 5 //

JavaScript Expressions //
An expression is a combination of values, variables, and operators, //
.which computes to a value

.The computation is called an evaluation //

:For example, 5 * 10 evaluates to 50 //

(document.write(5 * 10 //

:Expressions can also contain variable values //
```

```
x * 10 //

.The values can be of various types, such as numbers and strings //

:"For example, "John" + " " + "Doe", evaluates to "John Doe //

("document.write("Daniel"+ " "+"Asa //

JavaScript Keywords //
.JavaScript keywords are used to identify actions to be performed //

:The let keyword tells the browser to create variables //

let x, y //
x = 5 + 6 //
y = x * 10 //

:The var keyword also tells the browser to create variables //

var x, y //
x = 5 + 6 //
y = x * 10 //

Note: In these examples, using var or let will produce the same //
.result. You will learn more about var and let later in this tutorial

JavaScript Comments //
."Not all JavaScript statements are "executed //

Code after double slashes // or between /* and */ is treated as a //
.comment

JavaScript Identifiers / Names //
.Identifiers are JavaScript names //

.Identifiers are used to name variables and keywords, and functions //

The rules for legal names are the same in most programming //
.languages

:A JavaScript name must begin with //
```

```
(A letter (A-Z or a-z //
($) A dollar sign //
(_) Or an underscore //
Subsequent characters may be letters, digits, underscores, or dollar //
signs

Note: Numbers are not allowed as the first character in names.This //
.way JavaScript can easily distinguish identifiers from numbers

JavaScript Variables //

:Ways to Declare a JavaScript Variable 4 //
Using var //
Using let //
Using const //
Using nothing //

?When to Use JavaScript var //
.Always declare JavaScript variables with var,let, orconst //

.The var keyword is used in all JavaScript code from 1995 to 2015 //

.The let and const keywords were added to JavaScript in 2015 //

.If you want your code to run in older browsers, you must use var //

?When to Use JavaScript const //
.If you want a general rule: always declare variables with const //

.If you think the value of the variable can change, use let //

In this example, price1, price2, and total, are variables //

JavaScript Identifiers //
.All JavaScript variables must be identified with unique names //

.These unique names are called identifiers //

Identifiers can be short names (like x and y) or more descriptive //
.(names (age, sum, totalVolume

The general rules for constructing names for variables (unique //
:identifiers) are
```

```
.Names can contain letters, digits, underscores, and dollar signs //
.Names must begin with a letter //
Names can also begin with $ and _ (but we will not use it in this //
.(tutorial
.(Names are case sensitive (y and Y are different variables //
.Reserved words (like JavaScript keywords) cannot be used as names //

The Assignment Operator //
In JavaScript, the equal sign (=) is an "assignment" operator, not //
.an "equal to" operator

This is different from algebra. The following does not make sense in //
:algebra

x = x + 5 //

In JavaScript, however, it makes perfect sense: it assigns the value //
.of x + 5 to x

It calculates the value of x + 5 and puts the result into x. The) //
(.value of x is incremented by 5

.Note: The "equal to" operator is written like == in JavaScript //

Value = undefined //
In computer programs, variables are often declared without a value. //
The value can be something that has to be calculated, or something that
.will be provided later, like user input

.A variable declared without a value will have the value undefined //

The variable carName will have the value undefined after the //
:execution of this statement

let name //

$ JavaScript Dollar Sign //
Since JavaScript treats a dollar sign as a letter, identifiers //
containing $ are valid variable names

;"let $ = "Hello World //
;let $$$ = 2 //
```

```
;let $myMoney = 5 //

(_) JavaScript Underscore //
Since JavaScript treats underscore as a letter, identifiers //
:containing _ are valid variable names

;"let _lastName = "Johnson //
;let _x = 2 //
;let _100 = 5 //

JavaScript Let //

.Variables defined with let can not be redeclared //

.You can not accidentally redeclare a variable declared with let //

Block Scope //
.Before ES6 (2015), JavaScript had Global Scope and Function Scope //

.ES6 introduced two important new JavaScript keywords: let and const //

.These two keywords provide Block Scope in JavaScript //

Variables declared inside a { } block cannot be accessed from //
:outside the block

} //
;let x = 2     //
{ //

.Variables declared with the var keyword can NOT have block scope //

Variables declared inside a { } block can be accessed from outside //
.the block

JavaScript Const //

.(The const keyword was introduced in ES6 (2015 //

.Variables defined with const cannot be Redeclared //

.Variables defined with const cannot be Reassigned //
```

```
Variables defined with const have Block Scope //

:A const variable cannot be reassigned //

const PI = 3.1456985234 //
PI = 3.14 //
PI = PI + 10 //

.Output: TypeError: Assignment to constant variable //

?When to use JavaScript const //
Always declare a variable with const when you know that the value //
.should not be changed

:Use const when you declare //

A new Array //
A new Object //
A new Function //
A new RegExp //

Constant Objects and Arrays //

;["const cars = ["Saab", "Volvo", "BMW //
;"cars[0] = "Toyota //
;("cars.push("Audi //

Output: Toyota,Volvo,BMW,Audi //

Constant Objects //

;{"const car = {type:"Fiat", model:"500", color:"white //
;"car.color = "red //
;"car.owner = "Johnson //

Output: Car owner is Johnson //

Redeclaring an existing var or let variable to const, in the same //
scope, is not allowed

JavaScript Operators //

let x = 10 //
```

```
JavaScript Addition //

let x = 5 //
let y = 5 //
let z = x + y //

JavaScript Multiplication //

let x = 5 //
let y = 5 //
let z = x * y //

JavaScript Arithmetic Operators //

Operator Description //
Addition  + //
Subtraction  - //
Multiplication  * //
(Exponentiation (ES2016 ** //
Division  / //
(Modulus (Division Remainder  % //
Increment ++ //
Decrement -- //

JavaScript Assignment Operators //

Operator Example Same As //
x = y x = y  = //
x += y  x = x + y =+ //
x -= y  x = x - y =- //
x *= y  x = x * y =* //
x /= y  x = x / y =/ //
x %= y  x = x % y =% //
x **= y x = x ** y  =** //

JavaScript Comparison Operators //

equal to == //
equal value and equal type  === //
not equal =! //
not equal value or not equal type  ==! //
greater than  < //
```

```
less than  > //
greater than or equal to =< //
less than or equal to => //
ternary operator  ? //

JavaScript String Comparison //
:All the comparison operators above can also be used on strings //

;"let text1 = "A //
;"let text2 = "B //
;let result = text1 < text2 //

JavaScript Logical Operators //
Operator Description //
logical and && //
logical or || //
logical not  ! //

JavaScript Assignment //

JavaScript Assignment Operators //
.Assignment operators assign values to JavaScript variables //

Operator Example Same As //
x = y x = y  = //
x += y  x = x + y =+ //
x -= y  x = x - y =- //
x *= y  x = x * y =* //
x /= y  x = x / y =/ //
x %= y  x = x % y =% //
x **= y x = x ** y  =** //

Shift Assignment Operators //
Operator Example Same As //
x <<= y x = x << y  =>> //
x >>= y x = x >> y  =<< //
x >>>= y  x = x >>> y =<<< //

Bitwise Assignment Operators //
Operator Example Same As //
x &= y  x = x & y =& //
x ^= y  x = x ^ y =^ //
x |= y  x = x | y =| //
```

```
Logical Assignment Operators //
Operator Example Same As //
(x &&= y x = x && (x = y  =&& //
(x ||= y x = x || (x = y  =|| //
(x ??= y x = x ?? (x = y  =?? //

JavaScript Data Types //

JavaScript has 8 Datatypes //
String .1 //
Number .2 //
Bigint .3 //
Boolean .4 //
Undefined .5 //
Null .6 //
Symbol .7 //
Object .8 //

The Object Datatype //
:The object data type can contain //

An object .1 //
An array .2 //
A date .3 //

:Numbers //
;let length = 16 //
;let weight = 7.5 //

:Strings //
;"let color = "Yellow //
;"let lastName = "Johnson //

Booleans //
;let x = true //
;let y = false //

:Object //
;{"const person = {firstName:"John", lastName:"Doe //

:Array object //
;["const cars = ["Saab", "Volvo", "BMW //
```

```
:Date object //
;("const date = new Date("2022-03-25 //

JavaScript Arrays //
.JavaScript arrays are written with square brackets //

.Array items are separated by commas //

The following code declares (creates) an array called cars, //
:(containing three items (car names

;["const cars = ["Saab", "Volvo", "BMW //

Array indexes are zero-based, which means the first item is [0], //
.second is [1], and so on

JavaScript Objects //
.{} JavaScript objects are written with curly braces //

Object properties are written as name:value pairs, separated by //
.commas

const person = {firstName:"John", lastName:"Doe", age:50, //
;{"eyeColor:"blue

The object (person) in the example above has 4 properties: //
.firstName, lastName, age, and eyeColor

The typeof Operator //
You can use the JavaScript typeof operator to find the type of a //
.JavaScript variable

:The typeof operator returns the type of a variable or an expression //

"typeof ""              // Returns "string //
"typeof "John"          // Returns "string //
"typeof "John Doe"      // Returns "string //

let x = true //

((console.log(typeof(x //
```

```
Output: bolian //

JavaScript Functions //

A JavaScript function is a block of code designed to perform a //
.particular task

A JavaScript function is executed when "something" invokes it (calls //
.(it

Function to compute the product of p1 and p2 //
} (function myFunction(p1, p2 //
;return p1 * p2     //
{   //

JavaScript Function Syntax //
A JavaScript function is defined with the function keyword, followed //
.() by a name, followed by parentheses

Function names can contain letters, digits, underscores, and dollar //
.(signs (same rules as variables

:The parentheses may include parameter names separated by commas //
(... ,parameter1, parameter2) //

The code to be executed, by the function, is placed inside curly //
{} :brackets

} (function name(parameter1, parameter2, parameter3 //
code to be executed //     //
{   //

Function parameters are listed inside the parentheses () in the //
.function definition

Function arguments are the values received by the function when it //
.is invoked

Inside the function, the arguments (the parameters) behave as local //
.variables

Function Return //
```

```
When JavaScript reaches a return statement, the function will stop //
.executing

If the function was invoked from a statement, JavaScript will //
."return" to execute the code after the invoking statement

Functions often compute a return value. The return value is //
:""returned" back to the "caller

let x = myFunction(4, 3);   // Function is called, return value will //
end up in x

} (function myFunction(a, b //
return a * b;              // Function returns the product of a and   //
b
{ //

output: 12 //

The () Operator Invokes the Function //
Using the example above, toCelsius refers to the function object, //
.and toCelsius() refers to the function result

Accessing a function without () will return the function object //
.instead of the function result

} (function toCelsius(fahrenheit //
;(return (5/9) * (fahrenheit-32     //
{   //

Note: In order for the function to be executed, at the end you must //
() call it with

JavaScript Objects //

This code assigns many values (Fiat, 500, white) to a variable named //
:car

;{"const car = {type:"Fiat", model:"500", color:"white //

} = const person //
,"firstName: "John     //
,"lastName : "Doe     //
```

```
,id       : 5566     //
} ()fullName : function     //
;return this.firstName + " " + this.lastName       //
{     //
;{   //

.In the example above, this refers to the person object //

.I.E. this.firstName means the firstName property of this //

.I.E. this.firstName means the firstName property of person //

.In JavaScript, the this keyword refers to an object //

.(Which object depends on how this is being invoked (used or called //

The this keyword refers to different objects depending on how it is //
:used

.In an object method, this refers to the object //
.Alone, this refers to the global object //
.In a function, this refers to the global object //
.In a function, in strict mode, this is undefined //
.In an event, this refers to the element that received the event //
Methods like call(), apply(), and bind() can refer this to any //
.object

Note: this is not a variable. It is a keyword. You cannot change the //
.value of this

!Do Not Declare Strings, Numbers, and Booleans as Objects //
When a JavaScript variable is declared with the keyword "new", the //
:variable is created as an object

x = new String();        // Declares x as a String object //
y = new Number();        // Declares y as a Number object //
z = new Boolean();       // Declares z as a Boolean object //

Avoid String, Number, and Boolean objects. They complicate your code //
.and slow down execution speed

JavaScript Events //
```

```
button onclick="document.getElementById('demo').innerHTML => */}
{/* <Date()">The time is?</button

In the example above, the JavaScript code changes the content of the //
."element with id="demo

In the next example, the code changes the content of its own element //
:((using this.innerHTML

{/* <button onclick="this.innerHTML = Date()">The time is?</button> */}

Common HTML Events //

:Here is a list of some common HTML events //

Event  Description //
onchange An HTML element has been changed //
onclick  The user clicks an HTML element //
onmouseover  The user moves the mouse over an HTML element //
onmouseout The user moves the mouse away from an HTML element //
onkeydown  The user pushes a keyboard key //
onload The browser has finished loading the page //

JavaScript Strings //

String Length //
:To find the length of a string, use the built-in length property //

;"let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ //
;let length = text.length //
(document.write(length //

Escape Character //

." The string will be chopped to "We are the so-called //

The solution to avoid this problem, is to use the backslash escape //
.character

The backslash (\) escape character turns special characters into //
:string characters

Code Result  Description //
```

```
Single quote ' '\ //
Double quote " "\ //
Backslash \ \\ //
:The sequence \"  inserts a double quote in a string //

;".let text = "We are the so-called \"Vikings\" from the north //

:Six other escape sequences are valid in JavaScript //

Code Result //
b Backspace\ //
f Form Feed\ //
n New Line\ //
r Carriage Return\ //
t Horizontal Tabulator\ //
v Vertical Tabulator\ //

JavaScript Strings as Objects //

:But strings can also be defined as objects with the keyword new //

;("let y = new String("John //

JavaScript String Methods //

String length //
()String slice //
()String substring //
()String substr //
()String replace //
()String replaceAll //
()String toUpperCase //
()String toLowerCase //
()String concat //
()String trim //
()String trimStart //
()String trimEnd //
()String padStart //
()String padEnd //
()String charAt //
()String charCodeAt //
()String split //
```

```
JavaScript String Length //

:The length property returns the length of a string //

;"let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ //
;let length = text.length //

Extracting String Parts //

:There are 3 methods for extracting a part of a string //

(slice(start, end //
(substring(start, end //
(substr(start, length //

()JavaScript String slice //
slice() extracts a part of a string and returns the extracted part //
.in a new string

The method takes 2 parameters: start position, and end position (end //
.(not included

;"let text = "Apple, Banana, Kiwi //
;(let part = text.slice(7, 13 //

()JavaScript String substring //
.()substring() is similar to slice //

The difference is that start and end values less than 0 are treated //
.()as 0 in substring

;"let str = "Apple, Banana, Kiwi //
;(let part = str.substring(7, 13 //

If you omit the second parameter, substring() will slice out the //
.rest of the string

()JavaScript String substr //
.()substr() is similar to slice //

The difference is that the second parameter specifies the length of //
.the extracted part
```

```
;"let str = "Apple, Banana, Kiwi //
;(let part = str.substr(7, 6 //

If you omit the second parameter, substr() will slice out the rest //
.of the string

;"let str = "Apple, Banana, Kiwi //
;(let part = str.substr(7 //

Replacing String Content //
The replace() method replaces a specified value with another value //
:in a string

;"!let text = "Please visit Microsoft //
;("let newText = text.replace("Microsoft", "W3Schools //

:By default, the replace() method replaces only the first match //

;"!let text = "Please visit Microsoft and Microsoft //
;("let newText = text.replace("Microsoft", "W3Schools //

()JavaScript String ReplaceAll //
:()In 2021, JavaScript introduced the string method replaceAll //

;("text = text.replaceAll("Cats","Dogs //
("text = text.replaceAll("cats","dogs //

Converting to Upper and Lower Case //
:()A string is converted to upper case with toUpperCase //

:()A string is converted to lower case with toLowerCase //

()JavaScript String toUpperCase //

;"!let text1 = "Hello World //
;()let text2 = text1.toUpperCase //

()JavaScript String toLowerCase //

let text1 = "Hello World!";       // String //
let text2 = text1.toLowerCase();  // text2 is text1 converted to //
lower
```

```
()JavaScript String concat //

:concat() joins two or more strings //

;"let text1 = "Hello //
;"let text2 = "World //
;(let text3 = text1.concat(" ", text2 //

The concat() method can be used instead of the plus operator. These //
:two lines do the same

;"!text = "Hello" + " " + "World //
;("!text = "Hello".concat(" ", "World //

()JavaScript String trim //
:The trim() method removes whitespace from both sides of a string //

;"      !let text1 = "      Hello World //
;()let text2 = text1.trim //

()JavaScript String trimStart //
.ECMAScript 2019 added the String method trimStart() to JavaScript //

The trimStart() method works like trim(), but removes whitespace //
.only from the start of a string

;"     !let text1 = "     Hello World //
;()let text2 = text1.trimStart //

()JavaScript String trimEnd //
.ECMAScript 2019 added the String method trimEnd() to JavaScript //

The trimEnd() method works like trim(), but removes whitespace only //
.from the end of a string

;"     !let text1 = "     Hello World //
;()let text2 = text1.trimEnd //

JavaScript String Padding //
ECMAScript 2017 added two String methods: padStart() and padEnd() to //
.support padding at the beginning and at the end of a string

()JavaScript String padStart //
```

```
:The padStart() method pads a string with another string //

;"let text = "5 //
;("let padded = text.padStart(4,"x //

()JavaScript String padEnd //
:The padEnd() method pads a string with another string //

;"let text = "5 //
;("let padded = text.padEnd(4,"x //

Extracting String Characters //
:There are 3 methods for extracting string characters //

(charAt(position //
(charCodeAt(position //
[ ] Property access //

()JavaScript String charAt //
The charAt() method returns the character at a specified index //
:(position) in a string

;"let text = "HELLO WORLD //
;(let char = text.charAt(0 //

()JavaScript String charCodeAt //
The charCodeAt() method returns the unicode of the character at a //
:specified index in a string

.(The method returns a UTF-16 code (an integer between 0 and 65535 //

;"let text = "HELLO WORLD //
;(let char = text.charCodeAt(0 //

Property Access //
:ECMAScript 5 (2009) allows property access [ ] on strings //

;"let text = "HELLO WORLD //
;[let char = text[0 //

()JavaScript String split //
:A string can be converted to an array with the split() method //
```

```
;"let text = "a,b,c,d,e,f //
;(",")const myArray = text.split //
[myArray[0 //

JavaScript String Search //

JavaScript Search Methods //
()String indexOf //
()String lastIndexOf //
()String search //
()String match //
()String matchAll //
()String includes //
()String startsWith //
()String endsWith //

()JavaScript String indexOf //
The indexOf() method returns the index of (position of) the first //
:occurrence of a string in a string

;"!let text = "Please locate where 'locate' occurs //
;("text.indexOf("locate //

()JavaScript String lastIndexOf //
The lastIndexOf() method returns the index of the last occurrence of //
:a specified text in a string

;"!let text = "Please locate where 'locate' occurs //
;("text.lastIndexOf("locate //

The lastIndexOf() methods searches backwards (from the end to the //
beginning), meaning: if the second parameter is 15, the search starts
at position 15, and searches to the beginning of the strin

;"!let text = "Please locate where 'locate' occurs //
;(text.lastIndexOf("locate", 15 //

()JavaScript String search //
The search() method searches a string for a string (or a regular //
:expression) and returns the position of the match

;"!let text = "Please locate where 'locate' occurs //
("text.search("locate //
```

```
()JavaScript String match //
The match() method returns an array containing the results of //
.(matching a string against a string (or a regular expression

;"let text = "The rain in SPAIN stays mainly in the plain //
;("text.match("ain //

()JavaScript String matchAll //
The matchAll() method returns an iterator containing the results of //
.(matching a string against a string (or a regular expression

let text = "I love cats. Cats are very easy to love. Cats are very //
".popular
;("const iterator = text.matchAll("Cats //

()JavaScript String includes //
The includes() method returns true if a string contains a specified //
.value

.Otherwise it returns false //

;".let text = "Hello world, welcome to the universe //
;("text.includes("world //

()JavaScript String startsWith //
The startsWith() method returns true if a string begins with a //
.specified value

:Otherwise it returns false //

;".let text = "Hello world, welcome to the universe //
;("text.startsWith("Hello //

()JavaScript String endsWith //
The endsWith() method returns true if a string ends with a specified //
.value

:Otherwise it returns false //

;"let text = "John Doe //
;("text.endsWith("Doe //
```

```
JavaScript Template Literals //

Back-Tics Syntax //
Template Literals use back-ticks (``) rather than the quotes ("") to //
:define a string

;`!let text = `Hello World //

Quotes Inside Strings //
With template literals, you can use both single and double quotes //
:inside a string

;`"let text = `He's often called "Johnny //

Multiline Strings //
:Template literals allows multiline strings //

= let text //
The quick` //
brown fox //
jumps over //
`the lazy dog //

Variable Substitutions //
:Template literals allow variables in strings //

;"let firstName = "John //
;"let lastName = "Doe //

;`!{let text = `Welcome ${firstName}, ${lastName //

Expression Substitution //
:Template literals allow expressions in strings //

;let price = 10 //
;let VAT = 0.25 //

{(let total = `Total: ${(price * (1 + VAT)).toFixed(2 //

HTML Templates //

;"let header = "Templates Literals //
;["let tags = ["template literals", "javascript", "es6 //
```

```
;`<let html = `<h2>${header}</h2><ul //
} (for (const x of tags //
;`<html += `<li>${x}</li   //
{ //

;`<html += `</ul //

JavaScript Numbers //

Floating Precision //

;let x = 0.2 + 0.1 //

;let x = (0.2 * 10 + 0.1 * 10) / 10 //

NaN - Not a Number //
NaN is a JavaScript reserved word indicating that a number is not a //
.legal number

Trying to do arithmetic with a non-numeric string will result in NaN //
:((Not a Number

;"let x = 100 / "Apple //


You can use the global JavaScript function isNaN() to find out if a //
:value is a not a number

;"let x = 100 / "Apple //
;(isNaN(x //

Infinity //
Infinity (or -Infinity) is the value JavaScript will return if you //
.calculate a number outside the largest possible number

;let myNumber = 2 //
Execute until Infinity // //
} (while (myNumber != Infinity //
;myNumber = myNumber * myNumber   //
{ //

:Division by 0 (zero) also generates Infinity //
```

```
;let x =  2 / 0 //
;let y = -2 / 0 //

.note: Infinity is a number: typeof Infinity returns number //

JavaScript BigInt //

JavaScript BigInt variables are used to store big integer values //
.that are too big to be represented by a normal JavaScript Number

JavaScript Integer Accuracy //
:JavaScript integers are only accurate up to 15 digits //

;let x = 999999999999999 //
;let y = 9999999999999999 //

.(BigInt is the second numeric data type in JavaScript (after Number //

With BigInt the total number of supported data types in JavaScript //
:is 8

String .1 //
Number .2 //
Bigint .3 //
Boolean .4 //
Undefined .5 //
Null .6 //
Symbol .7 //
Object .8 //

JavaScript Number Methods //

JavaScript Number Methods //
:These number methods can be used on all JavaScript numbers //

Method Description //
toString() Returns a number as a string //
toExponential()  Returns a number written in exponential notation //
toFixed()  Returns a number written with a number of decimals //
toPrecision()  Returns a number written with a specified length //
ValueOf()  Returns a number as a number //
```

```
The toString() Method //
.The toString() method returns a number as a string //

All number methods can be used on any type of numbers (literals, //
:(variables, or expressions

;let x = 123 //
;()x.toString //
;()toString.(123) //
;()toString.(23 + 100) //

The toExponential() Method //
toExponential() returns a string, with a number rounded and written //
.using exponential notation

A parameter defines the number of characters behind the decimal poi //

;let x = 9.656 //
;(x.toExponential(2 //
;(x.toExponential(4 //
;(x.toExponential(6 //

The toFixed() Method //
toFixed() returns a string, with the number written with a specified //
:number of decimals

;let x = 9.656 //
;(x.toFixed(0 //
;(x.toFixed(2 //
;(x.toFixed(4 //
;(x.toFixed(6 //

The toPrecision() Method //
toPrecision() returns a string, with a number written with a //
:specified length

;let x = 9.656 //
;()x.toPrecision //
;(x.toPrecision(2 //
;(x.toPrecision(4 //
;(x.toPrecision(6 //

The valueOf() Method //
```

```
.valueOf() returns a number as a number //

;let x = 123 //
;()x.valueOf //
;()valueOf.(123) //
;()valueOf.(23 + 100) //

In JavaScript, a number can be a primitive value (typeof = number) //
.(or an object (typeof = object

The valueOf() method is used internally in JavaScript to convert //
.Number objects to primitive values

There is no reason to use it in your code //

Node: All JavaScript data types have a valueOf() and a toString() //
.method

Converting Variables to Numbers //
There are 3 JavaScript methods that can be used to convert a //
:variable to a number

Method Description //
.Number() Returns a number converted from its argument //
parseFloat() Parses its argument and returns a floating point number //
parseInt() Parses its argument and returns a whole number //

The Number() Method //
The Number() method can be used to convert JavaScript variables to //
:numbers

;(Number(true //
;(Number(false //
;("Number("10 //
;("Number("  10 //
;("  Number("10 //
;("  Number(" 10 //
;("Number("10.33 //
;("Number("10,33 //
;("Number("10 33 //
;("Number("John //

The Number() Method Used on Dates //
```

```
.Number() can also convert a date to a number //

(("Number(new Date("1970-01-01 //

The parseInt() Method //
parseInt() parses a string and returns a whole number. Spaces are //
:allowed. Only the first number is returned

;("parseInt("-10 //
;("parseInt("-10.33 //
;("parseInt("10 //
;("parseInt("10.33 //
;("parseInt("10 20 30 //
;("parseInt("10 years //
;("parseInt("years 10 //

The parseFloat() Method //
parseFloat() parses a string and returns a number. Spaces are //
:allowed. Only the first number is returned

;("parseFloat("10 //
;("parseFloat("10.33 //
;("parseFloat("10 20 30 //
;("parseFloat("10 years //
;("parseFloat("years 10 //

Number Object Methods //
:These object methods belong to the Number object //

Method Description //
Number.isInteger() Returns true if the argument is an integer //
Number.isSafeInteger() Returns true if the argument is a safe //
integer
Number.parseFloat()  Converts a string to a number //
Number.parseInt()  Converts a string to a whole number //

Number Methods Cannot be Used on Variables //
.The number methods above belong to the JavaScript Number Object //

.()These methods can only be accessed like Number.isInteger //

:Using X.isInteger() where X is a variable, will result in an error //
```

```
.TypeError X.isInteger is not a function //

The Number.isInteger() Method //
The Number.isInteger() method returns true if the argument is an //
.integer

;(Number.isInteger(10 //
;(Number.isInteger(10.5 //

The Number.isSafeInteger() Method //
A safe integer is an integer that can be exactly represented as a //
.double precision number

The Number.isSafeInteger() method returns true if the argument is a //
.safe integer

;(Number.isSafeInteger(10 //
;(Number.isSafeInteger(12345678901234567890 //

The Number.parseFloat() Method //
.Number.parseFloat() parses a string and returns a number //

:Spaces are allowed. Only the first number is returned //

;("Number.parseFloat("10 //
;("Number.parseFloat("10.33 //
;("Number.parseFloat("10 20 30 //
;("Number.parseFloat("10 years //
;("Number.parseFloat("years 10 //

The Number.parseInt() Method //
.Number.parseInt() parses a string and returns a whole number //

:Spaces are allowed. Only the first number is returned //

;("Number.parseInt("-10 //
;("Number.parseInt("-10.33 //
;("Number.parseInt("10 //
;("Number.parseInt("10.33 //
;("Number.parseInt("10 20 30 //
;("Number.parseInt("10 years //
;("Number.parseInt("years 10 //
```

```
JavaScript Number Properties //


Property Description //
.EPSILON  The difference between 1 and the smallest number > 1 //
MAX_VALUE  The largest number possible in JavaScript //
MIN_VALUE  The smallest number possible in JavaScript //
(MAX_SAFE_INTEGER The maximum safe integer (253 - 1 //
(MIN_SAFE_INTEGER The minimum safe integer -(253 - 1 //
(POSITIVE_INFINITY  Infinity (returned on overflow //
(NEGATIVE_INFINITY  Negative infinity (returned on overflow //
NaN  A "Not-a-Number" value //

JavaScript EPSILON //
Number.EPSILON is the difference between 1 and the smallest floating //
.point number greater than 1

;let x = Number.EPSILON //

JavaScript MAX_VALUE //
Number.MAX_VALUE is a constant representing the largest possible //
.number in JavaScript

;let x = Number.MAX_VALUE //

Number Properties Cannot be Used on Variables //
.Number properties belong to the JavaScript Number Object //

.These properties can only be accessed as Number.MAX_VALUE //

Using x.MAX_VALUE, where x is a variable or a value, will return //
:undefined

;let x = 6 //
x.MAX_VALUE //

JavaScript MIN_VALUE //
Number.MIN_VALUE is a constant representing the lowest possible //
.number in JavaScript

;let x = Number.MIN_VALUE //

JavaScript MAX_SAFE_INTEGER //
```

```
Number.MAX_SAFE_INTEGER represents the maximum safe integer in //
.JavaScript

.(Number.MAX_SAFE_INTEGER is (253 - 1 //

;let x = Number.MAX_SAFE_INTEGER //

JavaScript MIN_SAFE_INTEGER //
Number.MIN_SAFE_INTEGER represents the minimum safe integer in //
.JavaScript

(Number.MIN_SAFE_INTEGER is -(253 - 1 //

;let x = Number.MIN_SAFE_INTEGER //

JavaScript Arrays //

:An array is a special variable, which can hold more than one value //

;["const cars = ["Saab", "Volvo", "BMW //

Creating an Array //
Using an array literal is the easiest way to create a JavaScript //
.Array

:Syntax //

;[... ,const array_name = [item1, item2 //

Using the JavaScript Keyword new //
The following example also creates an Array, and assigns values to //
:it

;("const cars = new Array("Saab", "Volvo", "BMW //

.()There is no need to use new Array //

For simplicity, readability and execution speed, use the array //
.literal method

Accessing Array Elements //
You access an array element by referring to the index number //
```

```
;["const cars = ["Saab", "Volvo", "BMW //
;[let car = cars[0 //

Changing an Array Element //
:This statement changes the value of the first element in cars //

;["const cars = ["Saab", "Volvo", "BMW //
;"cars[0] = "Opel //

Arrays are Objects //
Arrays are a special type of objects. The typeof operator in //
.JavaScript returns "object" for arrays

.But, JavaScript arrays are best described as arrays //

Arrays use numbers to access its "elements". In this example, //
:person[0] returns John

;[const person = ["John", "Doe", 46 //

Objects use names to access its "members". In this example, //
:person.firstName returns John

;{const person = {firstName:"John", lastName:"Doe", age:46 //
;(console.log(person.firstName //

The length Property //
The length property of an array returns the length of an array (the //
.(number of array elements

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;let length = fruits.length //

Looping Array Elements //
:One way to loop through an array, is using a for loop //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;let fLen = fruits.length //

;"<let text = "<ul //
} (++for (let i = 0; i < fLen; i //
;"<text += "<li>" + fruits[i] + "</li   //
{ //
```

```
;"<text += "</ul //

:You can also use the Array.forEach() function //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //

;"<let text = "<ul //
;(fruits.forEach(myFunction //
;"<text += "</ul //

} (function myFunction(value //
;"<text += "<li>" + value + "</li   //
{ //

Adding Array Elements //
The easiest way to add a new element to an array is using the push() //
:method

;["const fruits = ["Banana", "Orange", "Apple //
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits //

()JavaScript new Array //
.()JavaScript has a built-in array constructor new Array //

.But you can safely use [] instead //

These two different statements both create a new empty array named //
:points

;()const points = new Array //
;[] = const points //

;["const fruits = ["Banana", "Orange", "Apple //
;let type = typeof fruits //

:Solution 1 //
To solve this problem ECMAScript 5 (JavaScript 2009) defined a new //
:()method Array.isArray

;["const fruits = ["Banana", "Orange", "Apple //
;(Array.isArray(fruits //

:Solution 2 //
```

```
The instanceof operator returns true if an object is created by a //
:given constructor

;["const fruits = ["Banana", "Orange", "Apple //

;fruits instanceof Array //

JavaScript Array Methods //

Converting Arrays to Strings //
The JavaScript method toString() converts an array to a string of //
.(comma separated) array values

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;(()document.write(fruits.toString //

.The join() method also joins all array elements into a string //

It behaves just like toString(), but in addition you can specify the //
:separator

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
((" * ")document.write(fruits.join //

Popping and Pushing //
When you work with arrays, it is easy to remove elements and add new //
.elements

:This is what popping and pushing is //

.Popping items out of an array, or pushing items into an array //

()JavaScript Array pop //
:The pop() method removes the last element from an array //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()fruits.pop //

()JavaScript Array push //
:(The push() method adds a new element to an array (at the end //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;("fruits.push("Kiwi //
```

```
()JavaScript Array shift //
The shift() method removes the first array element and "shifts" all //
.other elements to a lower index

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()fruits.shift //

()JavaScript Array unshift //
The unshift() method adds a new element to an array (at the //
:beginning), and "unshifts" older elements

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;("fruits.unshift("Lemon //

! Warning //
.Array elements can be deleted using the JavaScript operator delete //

.Using delete leaves undefined holes in the array //

.Use pop() or shift() instead //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;[delete fruits[0 //

Merging (Concatenating) Arrays //
The concat() method creates a new array by merging (concatenating) //
:existing arrays

;["const myGirls = ["Cecilie", "Lone //
;["const myBoys = ["Emil", "Tobias", "Linus //

;(const myChildren = myGirls.concat(myBoys //

The concat() method does not change the existing arrays. It always //
.returns a new array

Flattening an Array //
Flattening an array is the process of reducing the dimensionality of //
.an array

The flat() method creates a new array with sub-array elements //
.concatenated to a specified depth
```

```
;[[const myArr = [[1,2],[3,4],[5,6 //
;()const newArr = myArr.flat //

Splicing and Slicing Arrays //
.The splice() method adds new items to an array //

The slice() method slices out a piece of an array //

()JavaScript Array splice //
:The splice() method can be used to add new items to an array //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;("fruits.splice(2, 0, "Lemon", "Kiwi //

:The splice() method returns an array with the deleted items //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;("fruits.splice(2, 2, "Lemon", "Kiwi //

Using splice() to Remove Elements //
With clever parameter setting, you can use splice() to remove //
:elements without leaving "holes" in the array

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;(fruits.splice(0, 1 //

()JavaScript Array slice //
.The slice() method slices out a piece of an array into a new array //

This example slices out a part of an array starting from array //
:("element 1 ("Orange

;["const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango //
;(const citrus = fruits.slice(1 //

Note //
.The slice() method creates a new array //

The slice() method does not remove any elements from the source //
.array
```

```
()Automatic toString //
JavaScript automatically converts an array to a comma separated //
.string when a primitive value is expected

.This is always the case when you try to output an array //

:These two examples will produce the same result //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()fruits.toString //

JavaScript Sorting Arrays //

:The sort() method sorts an array alphabetically //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()fruits.sort //

Reversing an Array //
.The reverse() method reverses the elements in an array //

:You can use it to sort an array in descending order //

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()fruits.sort //
;()fruits.reverse //

Numeric Sort //
.By default, the sort() function sorts values as strings //

."(This works well for strings ("Apple" comes before "Banana //

However, if numbers are sorted as strings, "25" is bigger than //
.""100", because "2" is bigger than "1

Because of this, the sort() method will produce incorrect result //
.when sorting numbers

:You can fix this by providing a compare function //

;[const points = [40, 100, 1, 5, 25, 10 //
;({points.sort(function(a, b){return a - b //
```

```
Sorting an Array in Random Order //

;[const points = [40, 100, 1, 5, 25, 10 //
;({()points.sort(function(){return 0.5 - Math.random //

;[const points = [40, 100, 1, 5, 25, 10 //

} (--for (let i = points.length -1; i > 0; i //
;((let j = Math.floor(Math.random() * (i+1   //
;[let k = points[i   //
;[points[i] = points[j   //
;points[j] = k   //
{ //

Using Math.max() on an Array //
:You can use Math.max.apply to find the highest number in an array //

;[const points = [40, 100, 1, 5, 25, 10 //
;(myArrayMax(points //
} (function myArrayMax(arr //
;(return Math.max.apply(null, arr     //
{   //

Using Math.min() on an Array //
:You can use Math.min.apply to find the lowest number in an array //

;[const points = [40, 100, 1, 5, 25, 10 //
;(myArrayMin(points //
} (function myArrayMin(arr //
;(return Math.min.apply(null, arr     //
{   //

Sorting Object Arrays //
:JavaScript arrays often contain objects //

] = const cars //
,{type:"Volvo", year:2016}     //
,{type:"Saab", year:2001}     //
{type:"BMW", year:2010}     //
;[   //

JavaScript Array Iteration //
```

```
.Array iteration methods operate on every array item //

()JavaScript Array forEach //
The forEach() method calls a function (a callback function) once for //
.each array element

;[const numbers = [45, 4, 9, 16, 25 //

;"" = let txt //
;(numbers.forEach(myFunction //
(document.writeln(txt //
} (function myFunction(value, index, array //
;"<txt += value + "<br     //
{   //

()JavaScript Array map //
The map() method creates a new array by performing a function on //
.each array element

The map() method does not execute the function for array elements //
.without values

.The map() method does not change the original array //

:This example multiplies each array value by 2 //

;[const numbers1 = [45, 4, 9, 16, 25 //
;(const numbers2 = numbers1.map(myFunction //
;(document.writeln(numbers2 //
} (function myFunction(value, index, array //
;return value * 2     //
{   //

()JavaScript Array flatMap //
.ES2019 added the Array flatMap() method to JavaScript //

The flatMap() method first maps all elements of an array and then //
.creates a new array by flattening the array

;[const myArr = [1, 2, 3, 4, 5, 6 //
;(const newArr = myArr.flatMap((x) => x * 2 //

()JavaScript Array filter //
```

```
The filter() method creates a new array with array elements that //
.pass a test

This example creates a new array from elements with a value larger //
:than 18

;[const numbers = [45, 4, 9, 16, 25 //
;(const over18 = numbers.filter(myFunction //
;(document.writeln(over18 //
} (function myFunction(value, index, array //
;return value > 18     //
{   //

()JavaScript Array reduce //
The reduce() method runs a function on each array element to produce //
.(reduce it to) a single value

The reduce() method works from left-to-right in the array. See also //
.()reduceRight

;[const numbers = [45, 4, 9, 16, 25 //
;(let sum = numbers.reduce(myFunction //

} (function myFunction(total, value, index, array //
;return total + value   //
{ //

()JavaScript Array reduceRight //
The reduceRight() method runs a function on each array element to //
.produce (reduce it to) a single value

The reduceRight() works from right-to-left in the array. See also //
.()reduce

;[const numbers = [45, 4, 9, 16, 25 //
;(let sum = numbers.reduceRight(myFunction //

} (function myFunction(total, value, index, array //
;return total + value   //
{ //

()JavaScript Array every //
.The every() method checks if all array values pass a test //
```

```
:This example checks if all array values are larger than 18 //

;[const numbers = [45, 4, 9, 16, 25 //
;(let allOver18 = numbers.every(myFunction //

} (function myFunction(value, index, array //
;return value > 18   //
{ //

()JavaScript Array some //
.The some() method checks if some array values pass a test //

:This example checks if some array values are larger than 18 //

;[const numbers = [45, 4, 9, 16, 25 //
;(let someOver18 = numbers.some(myFunction //

} (function myFunction(value, index, array //
;return value > 18   //
{ //

()JavaScript Array indexOf //
The indexOf() method searches an array for an element value and //
.returns its position

;["const fruits = ["Apple", "Orange", "Apple", "Mango //
;let position = fruits.indexOf("Apple") + 1 //

Syntax //
(array.indexOf(item, start //


.item Required. The item to search for //
start  Optional. Where to start the search. Negative values will //
start at the given position counting from the end, and search to the
.end
.Array.indexOf() returns -1 if the item is not found //

If the item is present more than once, it returns the position of //
.the first occurrence

()JavaScript Array find //
```

```
The find() method returns the value of the first array element that //
.passes a test function

This example finds (returns the value of) the first element that is //
:larger than 18

;[const numbers = [4, 9, 16, 25, 29 //
;(let first = numbers.find(myFunction //

} (function myFunction(value, index, array //
;return value > 18   //
{ //

()JavaScript Array findIndex //
The findIndex() method returns the index of the first array element //
.that passes a test function

This example finds the index of the first element that is larger //
:than 18

;[const numbers = [4, 9, 16, 25, 29 //
;(let first = numbers.findIndex(myFunction //

} (function myFunction(value, index, array //
;return value > 18   //
{ //

()JavaScript Array Keys //
The Array.keys() method returns an Array Iterator object with the //
.keys of an array

;["const fruits = ["Banana", "Orange", "Apple", "Mango //
;()const keys = fruits.keys //

} (for (let x of keys //
;"<text += x + "<br   //
{ //

()JavaScript Array includes //
ECMAScript 2016 introduced Array.includes() to arrays. This allows //
us to check if an element is present in an array (including NaN, unlike
.(indexOf
```

```
;["const fruits = ["Banana", "Orange", "Apple", "Mango //

fruits.includes("Mango"); // is true //

Syntax //
(array.includes(search-item //

(...) JavaScript Array Spread //
The ... operator expands an iterable (like an array) into more //
:elements

;["const q1 = ["Jan", "Feb", "Mar //
;["const q2 = ["Apr", "May", "Jun //
;["const q3 = ["Jul", "Aug", "Sep //
;["const q4 = ["Oct", "Nov", "May //

;[const year = [...q1, ...q2, ...q3, ...q4 //

JavaScript Array Const //

(ECMAScript 2015 (ES6 //
.In 2015, JavaScript introduced an important new keyword: const //

:It has become a common practice to declare arrays using const //

;["const cars = ["Saab", "Volvo", "BMW //

Cannot be Reassigned //
:An array declared with const cannot be reassigned //

;["const cars = ["Saab", "Volvo", "BMW //
cars = ["Toyota", "Volvo", "Audi"];    // ERROR //

JavaScript Date Objects //

;()const d = new Date //

()JavaScript new Date //
:new Date() creates a date object with the current date and time //

;()const d = new Date //

JavaScript Date Formats //
```

```
JavaScript Date Input //
:There are generally 3 types of JavaScript date input formats //

Type Example //
(ISO Date "2015-03-25" (The International Standard //
"Short Date "03/25/2015 //
"Long Date  "Mar 25 2015" or "25 Mar 2015 //

Date Input - Parsing Dates //
If you have a valid date string, you can use the Date.parse() method //
.to convert it to milliseconds

Date.parse() returns the number of milliseconds between the date and //
:January 1, 1970

;("let msec = Date.parse("March 21, 2012 //

The new Date() Constructor //
.()In JavaScript, date objects are created with new Date //

.new Date() returns a date object with the current date and time //

;()const date = new Date //

Date Get Methods //
Method Description //
(getFullYear()  Get year as a four digit number (yyyy //
(getMonth() Get month as a number (0-11 //
(getDate()  Get day as a number (1-31 //
(getDay() Get weekday as a number (0-6 //
(getHours() Get hour (0-23 //
(getMinutes() Get minute (0-59 //
(getSeconds() Get second (0-59 //
(getMilliseconds()  Get millisecond (0-999 //
(getTime()  Get time (milliseconds since January 1, 1970 //

The getFullYear() Method //
The getFullYear() method returns the year of a date as a four digit //
:number

;("const d = new Date("2021-03-25 //
;()d.getFullYear //
```

```
The getMonth() Method //
The getMonth() method returns the month of a date as a number //
.((0-11

;("const d = new Date("2021-03-25 //
;()d.getMonth //

const months = ["January", "February", "March", "April", "May", //
"June", "July", "August", "September", "October", "November",
;[""December

;("const d = new Date("2021-03-25 //
;[()let month = months[d.getMonth //

The getDate() Method //
:(The getDate() method returns the day of a date as a number (1-31 //

;("const d = new Date("2021-03-25 //
;()d.getDate //

The getHours() Method //
The getHours() method returns the hours of a date as a number //
:((0-23

;("const d = new Date("2021-03-25 //
;()d.getHours //

The getMinutes() Method //
The getMinutes() method returns the minutes of a date as a number //
:((0-59

;("const d = new Date("2021-03-25 //
;()d.getMinutes //

The getSeconds() Method //
The getSeconds() method returns the seconds of a date as a number //
:((0-59

;("const d = new Date("2021-03-25 //
;()d.getSeconds //

The getMilliseconds() Method //
```

```
The getMilliseconds() method returns the milliseconds of a date as a //
:(number (0-999

;("const d = new Date("2021-03-25 //
;()d.getMilliseconds //

The getDay() Method //
.(The getDay() method returns the weekday of a date as a number (0-6 //

;("const d = new Date("2021-03-25 //
;()d.getDay //

const days = ["Sunday", "Monday", "Tuesday", "Wednesday", //
;[""Thursday", "Friday", "Saturday

;("const d = new Date("2021-03-25 //
;[()let day = days[d.getDay //

The getTime() Method //
The getTime() method returns the number of milliseconds since //
:January 1, 1970

;("const d = new Date("1970-01-01 //
;()d.getTime //

The Date.now() Method //
.Date.now() returns the number of milliseconds since January 1, 1970 //

;()let ms = Date.now //

:Calculate the number of years since 1970/01/01 //

;const minute = 1000 * 60 //
;const hour = minute * 60 //
;const day = hour * 24 //
;const year = day * 365 //

;(let years = Math.round(Date.now() / year //

UTC Date Get Methods //
Method Same As Description //
getUTCDate() getDate() Returns the UTC date //
getUTCFullYear() getFullYear() Returns the UTC year //
```

```
getUTCMonth()  getMonth()  Returns the UTC month //
getUTCDay()  getDay()  Returns the UTC day //
getUTCHours()  getHours()  Returns the UTC hour //
getUTCMinutes()  getMinutes()  Returns the UTC minutes //
getUTCSeconds()  getSeconds()  Returns the UTC seconds //
getUTCMilliseconds() getMilliseconds() Returns the UTC milliseconds //

The getTimezoneOffset() Method //
The getTimezoneOffset() method returns the difference (in minutes) //
:between local time an UTC time

;()let diff = d.getTimezoneOffset //

JavaScript Set Date Methods //

Set Date methods let you set date values (years, months, days, //
.hours, minutes, seconds, milliseconds) for a Date Object

Set Date Methods //
:Set Date methods are used for setting a part of a date //

Method Description //
(setDate()  Set the day as a number (1-31 //
(setFullYear()  Set the year (optionally month and day //
(setHours() Set the hour (0-23 //
(setMilliseconds()  Set the milliseconds (0-999 //
(setMinutes() Set the minutes (0-59 //
(setMonth() Set the month (0-11 //
(setSeconds() Set the seconds (0-59 //
(setTime()  Set the time (milliseconds since January 1, 1970 //

The setFullYear() Method //
The setFullYear() method sets the year of a date object. In this //
:example to 2020

;()const d = new Date //
;(d.setFullYear(2020 //

:The setFullYear() method can optionally set month and day //

;()const d = new Date //
;(d.setFullYear(2020, 11, 3 //
```

```
The setMonth() Method //
:(The setMonth() method sets the month of a date object (0-11 //

;()const d = new Date //
;(d.setMonth(11 //

The setDate() Method //
:(The setDate() method sets the day of a date object (1-31 //

;()const d = new Date //
;(d.setDate(15 //

:The setDate() method can also be used to add days to a date //

;()const d = new Date //
;(d.setDate(d.getDate() + 50 //

The setHours() Method //
:(The setHours() method sets the hours of a date object (0-23 //

;()const d = new Date //
;(d.setHours(22 //

The setMinutes() Method //
:(The setMinutes() method sets the minutes of a date object (0-59 //

;()const d = new Date //
;(d.setMinutes(30 //

The setSeconds() Method //
:(The setSeconds() method sets the seconds of a date object (0-59 //

;()const d = new Date //
;(d.setSeconds(30 //

JavaScript Math Object //

The JavaScript Math object allows you to perform mathematical tasks //
.on numbers

;(console.log(Math.PI //

(Math Properties (Constants //
```

```
.The syntax for any Math property is : Math.property //

JavaScript provides 8 mathematical constants that can be accessed as //
:Math properties

Math.E        // returns Euler's number //
Math.PI       // returns PI //
Math.SQRT2    // returns the square root of 2 //
Math.SQRT1_2  // returns the square root of 1/2 //
Math.LN2      // returns the natural logarithm of 2 //
Math.LN10     // returns the natural logarithm of 10 //
Math.LOG2E    // returns base 2 logarithm of E //
Math.LOG10E   // returns base 10 logarithm of E //

Math Methods //
(The syntax for Math any methods is : Math.method(number //

Number to Integer //
:There are 4 common methods to round a number to an integer //

Math.round(x)  Returns x rounded to its nearest integer //
Math.ceil(x) Returns x rounded up to its nearest integer //
Math.floor(x)  Returns x rounded down to its nearest integer //
(Math.trunc(x)  Returns the integer part of x (new in ES6 //

()Math.round //
:Math.round(x) returns the nearest integer //

;(Math.round(4.6 //

()Math.ceil //
Math.ceil(x) returns the value of x rounded up to its nearest //
:integer

;(Math.ceil(4.9 //
;(Math.ceil(4.7 //
;(Math.ceil(4.4 //
;(Math.ceil(4.2 //
;(Math.ceil(-4.2 //

()Math.floor //
Math.floor(x) returns the value of x rounded down to its nearest //
:integer
```

```
;(Math.floor(4.9 //
;(Math.floor(4.7 //
;(Math.floor(4.4 //
;(Math.floor(4.2 //
;(Math.floor(-4.2 //

()Math.trunc //
:Math.trunc(x) returns the integer part of x //

;(Math.trunc(4.9 //
;(Math.trunc(4.7 //
;(Math.trunc(4.4 //
;(Math.trunc(4.2 //
;(Math.trunc(-4.2 //

()Math.sign //
Math.sign(x) returns if x is negative, null or positive //

;(Math.sign(-4 //
;(Math.sign(0 //
;(Math.sign(4 //

()Math.pow //
:Math.pow(x, y) returns the value of x to the power of y //

;(Math.pow(8, 2 //

()Math.sqrt //
:Math.sqrt(x) returns the square root of x //

;(Math.sqrt(64 //

()Math.abs //
:Math.abs(x) returns the absolute (positive) value of x //

;(Math.abs(-4.7 //

()Math.sin //
Math.sin(x) returns the sine (a value between -1 and 1) of the angle //
.(x (given in radians
```

```
If you want to use degrees instead of radians, you have to convert //
:degrees to radians

.Angle in radians = Angle in degrees x PI / 180 //

Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 //
(degrees

()Math.min() and Math.max //
Math.min() and Math.max() can be used to find the lowest or highest //
:value in a list of arguments

;(Math.min(0, 150, 30, 20, -8, -200 //

;(Math.max(0, 150, 30, 20, -8, -200 //

()Math.random //
Math.random() returns a random number between 0 (inclusive), and 1 //
:((exclusive

;()Math.random //

The Math.log() Method //
.Math.log(x) returns the natural logarithm of x //

The natural logarithm returns the time needed to reach a certain //
:level of growth

;(Math.log(1 //

The Math.log2() Method //
.Math.log2(x) returns the base 2 logarithm of x //

;(Math.log2(8 //

The Math.log10() Method //
.Math.log10(x) returns the base 10 logarithm of x //

;(Math.log10(1000 //

JavaScript Math Methods //
Method Description //
abs(x) Returns the absolute value of x //
```

```
acos(x)  Returns the arccosine of x, in radians //
acosh(x) Returns the hyperbolic arccosine of x //
asin(x)  Returns the arcsine of x, in radians //
asinh(x) Returns the hyperbolic arcsine of x //
atan(x)  Returns the arctangent of x as a numeric value between //
-PI/2 and PI/2 radians
atan2(y, x)  Returns the arctangent of the quotient of its arguments //
atanh(x) Returns the hyperbolic arctangent of x //
cbrt(x)  Returns the cubic root of x //
ceil(x)  Returns x, rounded upwards to the nearest integer //
(cos(x) Returns the cosine of x (x is in radians //
cosh(x)  Returns the hyperbolic cosine of x //
exp(x) Returns the value of Ex //
floor(x) Returns x, rounded downwards to the nearest integer //
log(x) Returns the natural logarithm (base E) of x //
max(x, y, z, ..., n) Returns the number with the highest value //
min(x, y, z, ..., n) Returns the number with the lowest value //
pow(x, y)  Returns the value of x to the power of y //
random() Returns a random number between 0 and 1 //
round(x) Rounds x to the nearest integer //
(sign(x)  Returns if x is negative, null or positive (-1, 0, 1 //
(sin(x) Returns the sine of x (x is in radians //
sinh(x)  Returns the hyperbolic sine of x //
sqrt(x)  Returns the square root of x //
tan(x) Returns the tangent of an angle //
tanh(x)  Returns the hyperbolic tangent of a number //
(trunc(x) Returns the integer part of a number (x //

JavaScript Random //

()Math.random //
Math.random() returns a random number between 0 (inclusive),  and 1 //
:((exclusive

:Returns a random number //
;()Math.random //

JavaScript Random Integers //
Math.random() used with Math.floor() can be used to return random //
.integers

:Returns a random integer from 0 to 9 //
;(Math.floor(Math.random() * 10 //
```

```
JavaScript Booleans //

.A JavaScript Boolean represents one of two values: true or false //

Boolean Values //
Very often, in programming, you will need a data type that can only //
have one of two values, like

YES / NO //
ON / OFF //
TRUE / FALSE //
For this, JavaScript has a Boolean data type. It can only take the //
.values true or false

The Boolean() Function //
You can use the Boolean() function to find out if an expression (or //
:a variable) is true

Boolean(10 > 9) // => true //

Comparisons and Conditions //
The chapter JS Comparisons gives a full overview of comparison //
.operators

The chapter JS Conditions gives a full overview of conditional //
.statements

:Here are some examples //

Operator Description Example //
("equal to  if (day == "Monday == //
(greater than  if (salary > 9000  < //
(less than if (age < 18  > //

Everything Without a "Value" is False //

:The Boolean value of 0 (zero) is false //

;let x = 0 //
;(Boolean(x //

:The Boolean value of -0 (minus zero) is false //
```

```
;let x = -0 //
;(Boolean(x //

:The Boolean value of "" (empty string) is false //

;"" = let x //
;(Boolean(x //

:The Boolean value of undefined is false //

;let x //
;(Boolean(x //

:The Boolean value of null is false //

;let x = null //
;(Boolean(x //

:The Boolean value of false is (you guessed it) false //

;let x = false //
;(Boolean(x //

:The Boolean value of NaN is false //

;"let x = 10 / "Hallo //
;(Boolean(x //

JavaScript Booleans as Objects //
Normally JavaScript booleans are primitive values created from //
literal

;let x = false //

:But booleans can also be defined as objects with the keyword new //

;(let y = new Boolean(false //

JavaScript Comparison and Logical Operators //

.Comparison and Logical operators are used to test for true or false //
```

```
Comparison Operators //
Comparison operators are used in logical statements to determine //
.equality or difference between variables or values

:Given that x = 5, the table below explains the comparison operators //

Operator Description Comparing Returns Try it //
equal to  x == 8  false == //
x == 5 true //
x == "5" true //
equal value and equal type  x === 5 true  === //
x === "5"  false //
not equal x != 8  true =! //
not equal value or not equal type x !== 5 false  ==! //
x !== "5"  true //
x !== 8  true //
greater than  x > 8 false  < //
less than x < 8 true  > //
greater than or equal to  x >= 8  false =< //
less than or equal to x <= 8  true => //

Logical Operators //
Logical operators are used to determine the logic between variables //
.or values

Given that x = 6 and y = 3, the table below explains the logical //
:operators

Operator Description Example Try it //
and (x < 10 && y > 1) is true && //
or  (x == 5 || y == 5) is false || //
not !(x == y) is true  ! //

Comparing Different Types //
.Comparing data of different types may give unexpected results //

When comparing a string with a number, JavaScript will convert the //
string to a number when doing the comparison. An empty string converts
.to 0. A non-numeric string converts to NaN which is always false

Case Value Try //
true 12 > 2 //
true "12" > 2 //
```

```
John" false" > 2 //
John" false" < 2 //
John"  false" == 2 //
false "12" > "2" //
true "12" < "2" //
false  "12" == "2" //

;(age = Number(age //
} ((if (isNaN(age //
;"voteable = "Input is not a number   //
} else { //
;"voteable = (age < 18) ? "Too young" : "Old enough   //
{ //

(??) The Nullish Coalescing Operator //
The ?? operator returns the first argument if it is not nullish //
.((null or undefined

.Otherwise it returns the second argument //

;let name = null //
;"let text = "missing //
;let result = name ?? text //

JavaScript if, else, and else if //

Conditional statements are used to perform different actions based //
.on different conditions

Conditional Statements //
Very often when you write code, you want to perform different //
.actions for different decisions

.You can use conditional statements in your code to do this //

:In JavaScript we have the following conditional statements //

Use if to specify a block of code to be executed, if a specified //
condition is true
Use else to specify a block of code to be executed, if the same //
condition is false
Use else if to specify a new condition to test, if the first //
condition is false
```

```
Use switch to specify many alternative blocks of code to be executed //

The if Statement //
Use the if statement to specify a block of JavaScript code to be //
.executed if a condition is true

Syntax //
} (if (condition //
block of code to be executed if the condition is true  //   //
{ //

:Make a "Good day" greeting if the hour is less than 18:00 //

} (if (hour < 18 //
;"greeting = "Good day   //
{ //

The else Statement //
Use the else statement to specify a block of code to be executed if //
.the condition is false

} (if (condition //
block of code to be executed if the condition is true  //   //
} else { //
block of code to be executed if the condition is false  //   //
{ //

If the hour is less than 18, create a "Good day" greeting, otherwise //
:""Good evening

} (if (hour < 18 //
;"greeting = "Good day   //
} else { //
;"greeting = "Good evening   //
{ //
:The result of greeting will be //

Good evening //

The else if Statement //
Use the else if statement to specify a new condition if the first //
.condition is false
```

```
Syntax //
} (if (condition1 //
block of code to be executed if condition1 is true  //   //
} (else if (condition2 { //
block of code to be executed if the condition1 is false and  //   //
condition2 is true
} else { //
block of code to be executed if the condition1 is false and  //   //
condition2 is false
{ //

If time is less than 10:00, create a "Good morning" greeting, if //
not, but time is less than 20:00, create a "Good day" greeting,
:"otherwise a "Good evening

} (if (time < 10 //
;"greeting = "Good morning   //
} (else if (time < 20 { //
;"greeting = "Good day   //
} else { //
;"greeting = "Good evening   //
{ //
:The result of greeting will be //

Good evening //

JavaScript Switch Statement //

The switch statement is used to perform different actions based on //
.different conditions

The JavaScript Switch Statement //
Use the switch statement to select one of many code blocks to be //
.executed

Syntax //
} (switch(expression //
:case x   //
code block //     //
;break     //
:case y   //
code block //     //
;break     //
```

```
:default   //
code block //     //
{ //

:This is how it works //

.The switch expression is evaluated once //
The value of the expression is compared with the values of each //
.case
.If there is a match, the associated block of code is executed //
.If there is no match, the default code block is executed //

.The getDay() method returns the weekday as a number between 0 and 6 //

(.. Sunday=0, Monday=1, Tuesday=2) //

:This example uses the weekday number to calculate the weekday name //

} (()switch (new Date().getDay //
:case 0   //
;"day = "Sunday     //
;break     //
:case 1   //
;"day = "Monday     //
;break     //
:case 2   //
;"day = "Tuesday      //
;break     //
:case 3   //
;"day = "Wednesday     //
;break     //
:case 4   //
;"day = "Thursday     //
;break     //
:case 5   //
;"day = "Friday     //
;break     //
:case 6   //
;"day = "Saturday     //
{ //
:The result of day will be //

Saturday //
```

```
The break Keyword //
When JavaScript reaches a break keyword, it breaks out of the switch //
.block

.This will stop the execution inside the switch block //

It is not necessary to break the last case in a switch block. The //
.block breaks (ends) there anyway

Note: If you omit the break statement, the next case will be //
.executed even if the evaluation does not match the case

.The getDay() method returns the weekday as a number between 0 and 6 //

If today is neither Saturday (6) nor Sunday (0), write a default //
:message

} (()switch (new Date().getDay //
:case 6   //
;"text = "Today is Saturday     //
;break     //
:case 0   //
;"text = "Today is Sunday     //
;break     //
:default   //
;"text = "Looking forward to the Weekend     //
{ //
:The result of text will be //

Today is Saturday //

The default case does not have to be the last case in a switch //
:block

Example //
} (()switch (new Date().getDay //
:default   //
;"text = "Looking forward to the Weekend     //
;break     //
:case 6   //
;"text = "Today is Saturday     //
;break     //
```

```
:case 0   //
;"text = "Today is Sunday     //
{ //

If default is not the last case in the switch block, remember to end //
.the default case with a break

Common Code Blocks //
.Sometimes you will want different switch cases to use the same code //

In this example case 4 and 5 share the same code block, and 0 and 6 //
:share another code block

} (()switch (new Date().getDay //
:case 4   //
:case 5   //
;"text = "Soon it is Weekend     //
;break     //
:case 0   //
:case 6   //
;"text = "It is Weekend     //
;break     //
:default   //
;"text = "Looking forward to the Weekend     //
{ //

JavaScript For Loop //

.Loops can execute a block of code a number of times //

JavaScript Loops //
Loops are handy, if you want to run the same code over and over //
.again, each time with a different value

:Often this is the case when working with arrays //

Different Kinds of Loops //
:JavaScript supports different kinds of loops //

for - loops through a block of code a number of times //
for/in - loops through the properties of an object //
for/of - loops through the values of an iterable object //
```

```
while - loops through a block of code while a specified condition is //
true
do/while - also loops through a block of code while a specified //
condition is true

The For Loop //
:The for statement creates a loop with 3 optional expressions //

} (for (expression 1; expression 2; expression 3 //
code block to be executed //   //
{ //
Expression 1 is executed (one time) before the execution of the code //
.block

.Expression 2 defines the condition for executing the code block //

Expression 3 is executed (every time) after the code block has been //
.executed

} (++for (let i = 0; i < 5; i //
;"<text += "The number is " + i + "<br     //
{   //

And you can omit expression 1 (like when your values are set before //
:(the loop starts

Example //
;let i = 2 //
;let len = cars.length //
;"" = let text //
} (++for (; i < len; i //
;"<text += cars[i] + "<br   //
{ //

Loop Scope //
:Using var in a loop //

;var i = 5 //

} (++for (var i = 0; i < 10; i //
some code //   //
{ //
```

```
Here i is 10 // //

:Using let in a loop //

;let i = 5 //

} (++for (let i = 0; i < 10; i //
some code //   //
{ //

Here i is 5 // //

In the first example, using var, the variable declared in the loop //
.redeclares the variable outside the loop

In the second example, using let, the variable declared in the loop //
.does not redeclare the variable outside the loop

When let is used to declare the i variable in a loop, the i variable //
.will only be visible within the loop

For/Of and For/In Loops //
The for/in loop and the for/of loop are explained in the next //
.chapter

While Loops //
.The while loop and the do/while are explained in the next chapters //

JavaScript For In //

The For In Loop //
The JavaScript for in statement loops through the properties of an //
:Object

Syntax //
} (for (key in object //
code block to be executed //   //
{ //

;{const person = {fname:"John", lname:"Doe", age:25 //

;"" = let text //
} (for (let x in person //
```

```
;[text += person[x   //
{ //

For In Over Arrays //
The JavaScript for in statement can also loop over the properties of //
:an Array

Syntax //
} (for (variable in array //
code   //
{ //

;[const numbers = [45, 4, 9, 16, 25 //

;"" = let txt //
} (for (let x in numbers //
;[txt += numbers[x   //
{ //

.Do not use for in over an Array if the index order is important //

The index order is implementation-dependent, and array values may //
.not be accessed in the order you expect

It is better to use a for loop, a for of loop, or Array.forEach() //
.when the order is important

()Array.forEach //
The forEach() method calls a function (a callback function) once for //
.each array element

Example //
;[const numbers = [45, 4, 9, 16, 25 //

;"" = let txt //
;(numbers.forEach(myFunction //

} (function myFunction(value, index, array //
;txt += value   //
{ //

:Note that the function takes 3 arguments //
```

```
The item value //
The item index //
The array itself //
The example above uses only the value parameter. It can be rewritten //
:to

;[const numbers = [45, 4, 9, 16, 25 //

;"" = let txt //
;(numbers.forEach(myFunction //

} (function myFunction(value //
;txt += value   //
{ //

JavaScript For Of //

The For Of Loop //
The JavaScript for of statement loops through the values of an //
.iterable object

It lets you loop over iterable data structures such as Arrays, //
:Strings, Maps, NodeLists, and more

Syntax //
} (for (variable of iterable //
code block to be executed //   //
{ //

variable - For every iteration the value of the next property is //
assigned to the variable. Variable can be declared with const, let, or
.var

.iterable - An object that has iterable properties //

Looping over an Array //

;["const cars = ["BMW", "Volvo", "Mini //

;"" = let text //
} (for (let x of cars //
;text += x   //
{ //
```

```
Looping over a String //

;"let language = "JavaScript //

;"" = let text //
} (for (let x of language //
;text += x //
{ //

The While Loop //
The while loop and the do/while loop are explained in the next //
.chapter

JavaScript While Loop //

Loops can execute a block of code as long as a specified condition //
.is true

The While Loop //
The while loop loops through a block of code as long as a specified //
.condition is true

Syntax //
} (while (condition //
code block to be executed //   //
{ //
Example //
In the following example, the code in the loop will run, over and //
:over again, as long as a variable (i) is less than 10

} (while (i < 10 //
;text += "The number is " + i     //
;++i     //
{   //

The Do While Loop //
The do while loop is a variant of the while loop. This loop will //
execute the code block once, before checking if the condition is true,
.then it will repeat the loop as long as the condition is true

Syntax //
} do //
```

```
code block to be executed //   //
{ //
;(while (condition //

The example below uses a do while loop. The loop will always be //
executed at least once, even if the condition is false, because the
:code block is executed before the condition is tested

} do //
;text += "The number is " + i     //
;++i     //
{   //
;(while (i < 10   //

Comparing For and While //
If you have read the previous chapter, about the for loop, you will //
discover that a while loop is much the same as a for loop, with
.statement 1 and statement 3 omitted

The loop in this example uses a for loop to collect the car names //
:from the cars array

;["const cars = ["BMW", "Volvo", "Saab", "Ford //
;let i = 0 //
;"" = let text //

} (;[for (;cars[i //
;[text += cars[i   //
;++i   //
{ //

The loop in this example uses a while loop to collect the car names //
:from the cars array

;["const cars = ["BMW", "Volvo", "Saab", "Ford //
;let i = 0 //
;"" = let text //

} ([while (cars[i //
;[text += cars[i   //
;++i   //
{ //
```

```
JavaScript Break and Continue //

.The break statement "jumps out" of a loop //

.The continue statement "jumps over" one iteration in the loop //

The Break Statement //
You have already seen the break statement used in an earlier chapter //
.of this tutorial. It was used to "jump out" of a switch() statement

:The break statement can also be used to jump out of a loop //

} (++for (let i = 0; i < 10; i //
{ ;if (i === 3) { break     //
;"<text += "The number is " + i + "<br     //
{   //

In the example above, the break statement ends the loop ("breaks" //
.the loop) when the loop counter (i) is 3

The Continue Statement //
The continue statement breaks one iteration (in the loop), if a //
specified condition occurs, and continues with the next iteration in
.the loop

:This example skips the value of 3 //

} (++for (let i = 0; i < 10; i //
{ ;if (i === 3) { continue   //
;"<text += "The number is " + i + "<br   //
{ //

The continue statement (with or without a label reference) can only //
.be used to skip one loop iteration

The break statement, without a label reference, can only be used to //
.jump out of a loop or a switch

With a label reference, the break statement can be used to jump out //
:of any code block

;["const cars = ["BMW", "Volvo", "Saab", "Ford //
} :list //
```

```
;"<text += cars[0] + "<br   //
;"<text += cars[1] + "<br   //
;break list   //
;"<text += cars[2] + "<br   //
;"<text += cars[3] + "<br   //
{ //

JavaScript Iterables //

.(Iterables are iterable objects (like Arrays //

.Iterables can be accessed with simple and efficient code //

Iterables can be iterated over with for..of loops //

The For Of Loop //
The JavaScript for..of statement loops through the elements of an //
.iterable object

Syntax //
} (for (variable of iterable //
code block to be executed //   //
{ //

Iterating //
.Iterating is easy to understand //

.It simply means looping over a sequence of elements //

:Here are some easy examples //

Iterating over a String //
Iterating over an Array //
Iterating Over a String //
:You can use a for..of loop to iterate over the elements of a string //

;"const name = "W3Schools //

} (for (const x of name //
code block to be executed //   //
{ //

Iterating Over an Array //
```

```
:You can use a for..of loop to iterate over the elements of an Array //

Example //
;["const letters = ["a","b","c //

} (for (const x of letters //
code block to be executed //   //
{ //

Iterating Over a Set //
:You can use a for..of loop to iterate over the elements of a Set //

;(["const letters = new Set(["a","b","c //

} (for (const x of letters //
code block to be executed //   //
{ //

Iterating Over a Map //
:You can use a for..of loop to iterate over the elements of a Map //

])const fruits = new Map //
,[apples", 500"]   //
,[bananas", 300"]   //
[oranges", 200"]   //
;([ //

} (for (const x of fruits //
code block to be executed //   //
{ //

JavaScript Sets //

.A JavaScript Set is a collection of unique values //

.Each value can only occur once in a Set //

Essential Set Methods //
Method Description //
new Set()  Creates a new Set //
add()  Adds a new element to the Set //
delete() Removes an element from a Set //
has()  Returns true if a value exists in the Set //
```

```
forEach()  Invokes a callback for each element in the Set //
values() Returns an iterator with all the values in a Set //
Property Description //
size Returns the number of elements in a Set //

How to Create a Set //
:You can create a JavaScript Set by //

()Passing an Array to new Set //
Create a new Set and use add() to add values //
Create a new Set and use add() to add variables //
The new Set() Method //
:Pass an Array to the new Set() constructor //

Create a Set //
;(["const letters = new Set(["a","b","c //

:Create a Set and add values //

Create a Set // //
;()const letters = new Set //

Add Values to the Set // //
;("letters.add("a //
;("letters.add("b //
;("letters.add("c //

:Create a Set and add variables //

Create a Set // //
;()const letters = new Set //

Create Variables // //
;"const a = "a //
;"const b = "b //
;"const c = "c //

Add Variables to the Set // //
;(letters.add(a //
;(letters.add(b //
;(letters.add(c //

Create a new Set //
```

```
;(["const letters = new Set(["a","b","c //

Add a new Element //
;("letters.add("d //
;("letters.add("e //

Display set.size //

;(console.log(letters.size //

The forEach() Method //
The forEach() method invokes (calls) a function for each Set //
:element

Create a Set //
;(["const letters = new Set(["a","b","c //

List all Elements //
;"" = let text //
} (letters.forEach (function(value //
;text += value   //
({ //

The values() Method //
The values() method returns a new iterator object containing all the //
:values in a Set

Create a Set //
;(["const letters = new Set(["a","b","c //
[letters.values()   // Returns [object Set Iterator //

:Now you can use the Iterator object to access the elements //

Create a Set //
;(["const letters = new Set(["a","b","c //

List all Elements //
;"" = let text //
} (()for (const x of letters.values //
;text += x   //
{ //

JavaScript Maps //
```

```
.A Map holds key-value pairs where the keys can be any datatype //

.A Map remembers the original insertion order of the keys //

Essential Map Methods //
Method Description //
new Map()  Creates a new Map //
set()  Sets the value for a key in a Map //
get()  Gets the value for a key in a Map //
delete() Removes a Map element specified by the key //
has()  Returns true if a key exists in a Map //
forEach()  Calls a function for each key/value pair in a Map //
entries()  Returns an iterator with the [key, value] pairs in a Map //
Property Description //
size Returns the number of elements in a Map //
How to Create a Map //
:You can create a JavaScript Map by //

()Passing an Array to new Map //
()Create a Map and use Map.set //
The new Map() Method //
You can create a Map by passing an Array to the new Map() //
:constructor

Example //

Create a Map //
])const fruits = new Map //
,[apples", 500"]     //
,[bananas", 300"]     //
[oranges", 200"]     //
;([   //

(("console.log(fruits.get("apples //

Create a Map //
;()const fruits = new Map //

Set Map Values //
;(fruits.set("apples", 500 //
;(fruits.set("bananas", 300 //
;(fruits.set("oranges", 200 //
```

```
(("console.log(fruits.get("apples //

:The set() method can also be used to change existing Map values //

;(fruits.set("apples", 200 //

The get() Method //
:The get() method gets the value of a key in a Map //

fruits.get("apples");    // Returns 500 //

Create a Map //
])const fruits = new Map //
,[apples", 500"]     //
,[bananas", 300"]     //
[oranges", 200"]     //
;([   //

(console.log(fruits.size //

The delete() Method //
:The delete() method removes a Map element //

Create a Map //
])const fruits = new Map //
,[apples", 500"]     //
,[bananas", 300"]     //
[oranges", 200"]     //
;([   //

Delete an Element //
;("fruits.delete("apples   //

(console.log(fruits.siz //

Create a Map //
])const fruits = new Map //
,[apples", 500"]     //
,[bananas", 300"]     //
[oranges", 200"]     //
;([   //
```

```
(("console.log(fruits.has("apples //

The has() Method //
:The has() method returns true if a key exists in a Map //

Create a Map //
])const fruits = new Map //
,[apples", 500"]     //
,[bananas", 300"]     //
[oranges", 200"]     //
;([   //

(("console.log(fruits.has("apples //
```

JavaScript Objects vs Maps //
:Differences between JavaScript Objects and Maps //
Map Object //
Directly iterable Not directly iterable Iterable //
Have a size property Do not have a size property Size //
Keys can be any datatype Keys must be Strings (or Symbols) Key Types //
Keys are ordered by insertion Keys are not well ordered Key Order //
Do not have default keys Have default keys Defaults //
The forEach() Method //
:The forEach() method calls a function for each key/value pair in a Map //

List all entries //
;"" = let text //
} (fruits.forEach (function(value, key //
;text += key + ' = ' + value //
({ //

The entries() Method //
:The entries() method returns an iterator object with the [key, values] in a Map //

List all entries //
;"" = let text //
} (()for (const x of fruits.entries //
;text += x //
{ //

JavaScript typeof //

:In JavaScript there are 5 different data types that can contain values //

string //

number //
boolean //
object //
function //
:There are 6 types of objects //

Object //
Date //
Array //
String //
Number //
Boolean //
:And 2 data types that cannot contain values //

null //
undefined //

The typeof Operator //
.You can use the typeof operator to find the data type of a JavaScript variable //

"typeof "John" // Returns "string //
"typeof 3.14 // Returns "number //
"typeof NaN // Returns "number //
"typeof false // Returns "boolean //
"typeof [1,2,3,4] // Returns "object //
"typeof {name:'John', age:34} // Returns "object //
"typeof new Date() // Returns "object //
"typeof function () {} // Returns "function //
* "typeof myCar // Returns "undefined //
"typeof null // Returns "object //

Primitive Data //
A primitive data value is a single simple data value with no additional properties and //
.methods

:The typeof operator can return one of these primitive types //

string //
number //
boolean //
undefined //

"typeof "John" // Returns "string //
"typeof 3.14 // Returns "number //
"typeof true // Returns "boolean //
"typeof false // Returns "boolean //
(typeof x // Returns "undefined" (if x has no value //

Complex Data //
:The typeof operator can return one of two complex types //

function //
object //
.The typeof operator returns "object" for objects, arrays, and null //

.The typeof operator does not return "object" for functions //

"typeof {name:'John', age:34} // Returns "object //
(typeof [1,2,3,4] // Returns "object" (not "array", see note below //
"typeof null // Returns "object //
"typeof function myFunc(){} // Returns "function //

The Data Type of typeof //
The typeofoperator is not a variable. It is an operator. Operators (+ - * /) do not have any //
.data type

.(But, the typeof operator always returns a string (containing the type of the operand //

The constructor Property //
.The constructor property returns the constructor function for all JavaScript variables //

{[John".constructor // Returns function String() {[native code" //
{[constructor // Returns function Number() {[native code.(3.14) //
{[false.constructor // Returns function Boolean() {[native code //
{[constructor // Returns function Array() {[native code.[1,2,3,4] //
{[name:'John',age:34}.constructor // Returns function Object() {[native code} //
{[new Date().constructor // Returns function Date() {[native code //
{[function () {}.constructor // Returns function Function(){[native code //

You can check the constructor property to find out if an object is an Array (contains the //
:("word "Array

} (function isArray(myArray //
;return myArray.constructor.toString().indexOf("Array") > -1 //
{ //

:Or even simpler, you can check if the object is an Array function //

} (function isArray(myArray //
;return myArray.constructor === Array //
{ //

You can check the constructor property to find out if an object is a Date (contains the word //
:(""Date

} (function isDate(myDate //

```
;return myDate.constructor.toString().indexOf("Date") > -1     //
{   //

:Or even simpler, you can check if the object is a Date function //

} (function isDate(myDate //
;return myDate.constructor === Date     //
{   //
```

Undefined //
In JavaScript, a variable without a value, has the value undefined. The type is also //
.undefined

```
let car;    // Value is undefined, type is undefined //
```

Any variable can be emptied, by setting the value to undefined. The type will also be //
.undefined

```
car = undefined;    // Value is undefined, type is undefined //
```

Empty Values //
.An empty value has nothing to do with undefined //

.An empty string has both a legal value and a type //

```
"let car = "";    // The value is "", the typeof is "string //
```

Null //
.In JavaScript null is "nothing". It is supposed to be something that doesn't exist //

.Unfortunately, in JavaScript, the data type of null is an object //

:You can empty an object by setting it to null //

```
;{"let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //
person = null;    // Now value is null, but type is still an object //
```

:You can also empty an object by setting it to undefined //

```
;{"let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //
person = undefined;   // Now both value and type is undefined //
```

Difference Between Undefined and Null //
:undefined and null are equal in value but different in type //

```
typeof undefined           // undefined //
typeof null                // object //
```

null === undefined // false //
null == undefined // true //

The instanceof Operator //
:The instanceof operator returns true if an object is an instance of the specified object //

;["const cars = ["Saab", "Volvo", "BMW //

;(cars instanceof Array) //
;(cars instanceof Object) //
;(cars instanceof String) //
;(cars instanceof Number) //
The void Operator //
The void operator evaluates an expression and returns undefined. This operator is often //
used to obtain the undefined primitive value, using "void(0)" (useful when evaluating an
.(expression without using the return value

<";(a href="javascript:void(0> */}
Useless link
<a/>

<";('a href="javascript:void(document.body.style.backgroundColor='red>
Click me to change the background color of body to red
{/* <a/>

JavaScript Type Conversion //

Converting Strings to Numbers //
Converting Numbers to Strings //
Converting Dates to Numbers //
Converting Numbers to Dates //
Converting Booleans to Numbers //
Converting Numbers to Booleans //
JavaScript Type Conversion //
:JavaScript variables can be converted to a new variable and another data type //

By the use of a JavaScript function //
Automatically by JavaScript itself //
Converting Strings to Numbers //
.The global method Number() converts a variable (or a value) into a number //

.(A numeric string (like "3.14") converts to a number (like 3.14 //

.An empty string (like "") converts to 0 //

.(A non numeric string (like "John") converts to NaN (Not a Number //

:These will convert //

("Number("3.14 //
(Number(Math.PI //
(" ")Number //
("")Number //
:These will not convert //

("Number("99 88 //
("Number("John //

Number Methods //
In the chapter Number Methods, you will find more methods that can be used to convert //
:strings to numbers

Description Method //
Returns a number, converted from its argument Number() //
Parses a string and returns a floating point number parseFloat() //
Parses a string and returns an integer parseInt() //
The Unary + Operator //
:The unary + operator can be used to convert a variable to a number //

```
let y = "5";      // y is a string //
let x = + y;      // x is a number //
```

If the variable cannot be converted, it will still become a number, but with the value NaN //
:((Not a Number

```
let y = "John";   // y is a string //
(let x = + y;      // x is a number (NaN //
```

Converting Numbers to Strings //
.The global method String() can convert numbers to strings //

:It can be used on any type of numbers, literals, variables, or expressions //

```
String(x)         // returns a string from a number variable x //
String(123)       // returns a string from a number literal 123 //
String(100 + 23)  // returns a string from a number from an expression //
```

.The Number method toString() does the same //

```
()x.toString //
()toString.(123) //
()toString.(23 + 100) //
```

More Methods //
In the chapter Number Methods, you will find more methods that can be used to convert //
:numbers to strings

Description Method //
Returns a string, with a number rounded and written using exponential toExponential() //
.notation
Returns a string, with a number rounded and written with a specified number toFixed() //
.of decimals
Returns a string, with a number written with a specified length toPrecision() //

Converting Dates to Numbers //
.The global method Number() can be used to convert dates to numbers //

;()d = new Date //
Number(d) // returns 1404568027739 //
.The date method getTime() does the same //

;()d = new Date //
d.getTime() // returns 1404568027739 //

Converting Dates to Strings //
.The global method String() can convert dates to strings //

"(String(Date()) // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time //
.The Date method toString() does the same //

Example //
Date().toString() // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight //
"(Time
In the chapter Date Methods, you will find more methods that can be used to convert dates //
:to strings

Description Method //
(Get the day as a number (1-31 getDate() //
(Get the weekday a number (0-6 getDay() //
(Get the four digit year (yyyygetFullYear() //
(Get the hour (0-23 getHours() //
(Get the milliseconds (0-999 getMilliseconds() //
(Get the minutes (0-59 getMinutes() //
(Get the month (0-11 getMonth() //
(Get the seconds (0-59 getSeconds() //
(Get the time (milliseconds since January 1, 1970 getTime() //
Converting Booleans to Numbers //
.The global method Number() can also convert booleans to numbers //

Number(false) // returns 0 //
Number(true) // returns 1 //
Converting Booleans to Strings //
.The global method String() can convert booleans to strings //

"String(false)   // returns "false //
"String(true)   // returns "true //
.The Boolean method toString() does the same //

"false.toString()   // returns "false //
"true.toString()   // returns "true //
Automatic Type Conversion //
When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a //
."right" type

:The result is not always what you expect //

null   // returns 5   because null is converted to 0 + 5 //
"null // returns "5null"   because null is converted to "null + "5" //
"returns "52"   because 2 is converted to "2 //   2 + "5" //
returns 3   because "5" is converted to 5 //   2 - "5" //
returns 10   because "5" and "2" are converted to 5 and 2 //   "2" * "5" //

Automatic String Conversion //
JavaScript automatically calls the variable's toString() function when you try to "output" an //
:object or a variable

;document.getElementById("demo").innerHTML = myVar //

"[if myVar = {name:"Fjohn"} // toString converts to "[object Object //
"if myVar = [1,2,3,4]   // toString converts to "1,2,3,4 //
"if myVar = new Date()   // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200 //
:Numbers and booleans are also converted, but this is not very visible //

"if myVar = 123   // toString converts to "123 //
"if myVar = true   // toString converts to "true //
"if myVar = false   // toString converts to "false //
JavaScript Type Conversion Table //
This table shows the result of converting different JavaScript values to Number, String, and //
:Boolean

Original //
Converted   Value //
Converted   to Number //
Converted   to String //
Try it   to Boolean //
    false "false"   0 false //
     true   "true"   1   true //
    false   "0"   0   0 //
     true   "1"   1   1 //
     true   "0"   0   "0" //
     true   "000"   0"000" //

true "1" 1 "1" //
false "NaN" NaN NaN //
true "Infinity" Infinity Infinity //
true "-Infinity" -Infinity Infinity- //
false "" 0 "" //
true "20" 20 "20" //
true "twenty" NaN twenty"" //
true "" 0 [ ] //
true "20" 20 [20] //
true "10,20" NaN [10,20] //
true "twenty" NaN twenty"]"] //
true "ten,twenty" NaN ten","twenty"]"] //
true "function(){}" NaN function(){} //
true "[object Object]" NaN { } //
false "null" 0 null //
false "undefined" NaN undefined //
.Values in quotes indicate string values //

.Red values indicate values (some) programmers might not expect //

JavaScript Bitwise Operations //

JavaScript Bitwise Operators //
Description Name Operator //
Sets each bit to 1 if both bits are 1 AND & //
Sets each bit to 1 if one of two bits is 1 OR | //
Sets each bit to 1 if only one of two bits is 1 XOR ^ //
Inverts all the bits NOT ~ //
Shifts left by pushing zeros in from the right and let the leftmost bits fall off Zero fill left shift >> //
Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off Signed right shift << //
Shifts right by pushing zeros in from the left, and let the rightmost bits fall off Zero fill right shift <<< //
Examples //
Result Same as Result Operation //
0001 0001 & 0101 1 1 & 5 //
0101 0001 | 0101 5 1 | 5 //
1010 0101~ 10 5 ~ //
1010 1 >> 0101 10 1 >> 5 //
0100 0001 ^ 0101 4 1 ^ 5 //
0010 1 << 0101 2 1 << 5 //
0010 1 <<< 0101 2 1 <<< 5 //
JavaScript Uses 32 bits Bitwise Operands //
JavaScript stores numbers as 64 bits floating point numbers, but all bitwise operations are //
.performed on 32 bits binary numbers

Before a bitwise operation is performed, JavaScript converts numbers to 32 bits signed // .integers

After the bitwise operation is performed, the result is converted back to 64 bits JavaScript // .numbers

JavaScript Bitwise AND //
.When a bitwise AND is performed on a pair of bits, it returns 1 if both bits are 1 //

:One bit example //
Result Operation //
0 0 & 0 //
0 1 & 0 //
0 0 & 1 //
1 1 & 1 //
:bits example 4 //
Result Operation //
0000 0000 & 1111 //
0001 0001 & 1111 //
0010 0010 & 1111 //
0100 0100 & 1111 //
JavaScript Bitwise OR //
:When a bitwise OR is performed on a pair of bits, it returns 1 if one of the bits is 1 //

:One bit example //
Result Operation //
0 0 | 0 //
1 1 | 0 //
1 0 | 1 //
1 1 | 1 //
:bits example 4 //
Result Operation //
1111 0000 | 1111 //
1111 0001 | 1111 //
1111 0010 | 1111 //
1111 0100 | 1111 //
JavaScript Bitwise XOR //
:When a bitwise XOR is performed on a pair of bits, it returns 1 if the bits are different //

:One bit example //
Result Operation //
0 0 ^ 0 //
1 1 ^ 0 //
1 0 ^ 1 //
0 1 ^ 1 //
:bits example 4 //
Result Operation //
1111 0000 ^ 1111 //

1110 0001 ^ 1111 //
1101 0010 ^ 1111 //
1011 0100 ^ 1111 //
(&) JavaScript Bitwise AND //
:Bitwise AND returns 1 only if both bits are 1 //

Binary Decimal //
00000000000000000000000000000101 5 //
00000000000000000000000000000001 1 //
(1) 00000000000000000000000000000001 1 & 5 //

;let x = 5 & 1 //

(|) JavaScript Bitwise OR //
:Bitwise OR returns 1 if one of the bits is 1 //

Binary Decimal //
00000000000000000000000000000101 5 //
00000000000000000000000000000001 1 //
(5) 00000000000000000000000000000101 1 | 5 //

;let x = 5 | 1 //

(^) JavaScript Bitwise XOR //
:Bitwise XOR returns 1 if the bits are different //

Binary Decimal //
00000000000000000000000000000101 5 //
00000000000000000000000000000001 1 //
(4) 00000000000000000000000000000100 1 ^ 5 //

;let x = 5 ^ 1 //

(~) JavaScript Bitwise NOT //
Binary Decimal //
00000000000000000000000000000101 5 //
(6-) 11111111111111111111111111111010 5~ //

;let x = ~5 //

(>>) JavaScript (Zero Fill) Bitwise Left Shift //
This is a zero fill left shift. One or more zero bits are pushed in from the right, and the //
:leftmost bits fall off

Binary Decimal //
00000000000000000000000000000101 5 //
(10) 00000000000000000000000000001010 1 >> 5 //

;let x = 5 << 1 //

(<<) Right Shift //
This is a sign preserving right shift. Copies of the leftmost bit are pushed in from the left, //
:and the rightmost bits fall off

Binary Decimal //
11111111111111111111111111111011 5- //
(3-) 11111111111111111111111111111101 1 << 5- //

;let x = -5 >> 1 //

(<<<) JavaScript (Zero Fill) Right Shift //
This is a zero fill right shift. One or more zero bits are pushed in from the left, and the //
:rightmost bits fall off

Binary Decimal //
00000000000000000000000000000101 5 //
(2) 00000000000000000000000000000010 1 <<< 5 //

;let x = 5 >>> 1 //

Binary Numbers //
:Binary numbers with only one bit set are easy to understand //

Decimal value Binary Representation //
1 00000000000000000000000000000001 //
2 00000000000000000000000000000010 //
4 00000000000000000000000000000100 //
8 00000000000000000000000000001000 //
16 00000000000000000000000000010000 //
32 00000000000000000000000000100000 //
64 00000000000000000000000001000000 //
:Setting a few more bits reveals the binary pattern //

Decimal value Binary Representation //
(1 + 4) 5 00000000000000000000000000000101 //
(1 + 4 + 8) 13 00000000000000000000000000001101 //
(1 + 4 + 8 + 32) 45 00000000000000000000000000101101 //
.JavaScript binary numbers are stored in two's complement format //

:This means that a negative number is the bitwise NOT of the number plus 1 //

Decimal value Binary Representation //
5 00000000000000000000000000000101 //
5- 11111111111111111111111111111011 //
6 00000000000000000000000000000110 //
6- 11111111111111111111111111111010 //

40 00000000000000000000000000101000 //
40- 11111111111111111111111111011000 //
Converting Decimal to Binary //

}(function dec2bin(dec //
;(return (dec >>> 0).toString(2 //
{ //

Converting Binary to Decimal //

}(function bin2dec(bin //
;(return parseInt(bin, 2).toString(10 //
{ //

JavaScript Regular Expressions //

.A regular expression is a sequence of characters that forms a search pattern //

.The search pattern can be used for text search and text replace operations //

?What Is a Regular Expression //
.A regular expression is a sequence of characters that forms a search pattern //

When you search for data in a text, you can use this search pattern to describe what you //
.are searching for

.A regular expression can be a single character, or a more complicated pattern //

Regular expressions can be used to perform all types of text search and text replace //
.operations

Syntax //
;pattern/modifiers/ //

;w3schools/i/ //

Using String Methods //
In JavaScript, regular expressions are often used with the two string methods: search() //
.()and replace

The search() method uses an expression to search for a match, and returns the position of //
.the match

.The replace() method returns a modified string where the pattern is replaced //

Using String search() With a String //
The search() method searches a string for a specified value and returns the position of the //
:match

:Use a string to do a search for "W3schools" in a string //

;"!let text = "Visit W3Schools //
;("let n = text.search("W3Schools //
:The result in n will be //

6 //

Using String search() With a Regular Expression //

:Use a regular expression to do a case-insensitive search for "w3schools" in a string //

;"let text = "Visit W3Schools //
;(let n = text.search(/w3schools/i //
:The result in n will be //

6 //

Using String replace() With a String //
:The replace() method replaces a specified value with another value in a string //

;"!let text = "Visit Microsoft //
;("let result = text.replace("Microsoft", "W3Schools //

Use String replace() With a Regular Expression //

:Use a case insensitive regular expression to replace Microsoft with W3Schools in a string //

;"!let text = "Visit Microsoft //
;("let result = text.replace(/microsoft/i, "W3Schools //
:The result in res will be //

!Visit W3Schools //

Regular Expression Modifiers //
:Modifiers can be used to perform case-insensitive more global searches //

Try it Description Modifier //
Perform case-insensitive matching i //
(Perform a global match (find all matches rather than stopping after the first match g //

Perform multiline matching m //

Regular Expression Patterns //
:Brackets are used to find a range of characters //

Try it Description Expression //

Find any of the characters between the brackets abc]] //
Find any of the digits between the brackets [0-9] //
| Find any of the alternatives separated with x|y)) //
:Metacharacters are characters with a special meaning //

Try it Description Metacharacter //
Find a digit d\ //
Find a whitespace character s\ //
Find a match at the beginning of a word like this: \bWORD, or at the end of a word b\ //
like this: WORD\b
Find the Unicode character specified by the hexadecimal number xxxx uxxxx\ //
:Quantifiers define quantities //

Try it Description Quantifier //
Matches any string that contains at least one n n+ //
Matches any string that contains zero or more occurrences of n n* //
Matches any string that contains zero or one occurrences of n n? //
Using the RegExp Object //
In JavaScript, the RegExp object is a regular expression object with predefined properties //
.and methods

()Using test //
.The test() method is a RegExp expression method //

.It searches a string for a pattern, and returns true or false, depending on the result //

:"The following example searches a string for the character "e //

;/const pattern = /e //
;("!pattern.test("The best things in life are free //
:Since there is an "e" in the string, the output of the code above will be //

true //

You don't have to put the regular expression in a variable first. The two lines above can be //
:shortened to one

;("!e/.test("The best things in life are free/ //

()Using exec //
.The exec() method is a RegExp expression method //

.It searches a string for a specified pattern, and returns the found text as an object //

.If no match is found, it returns an empty (null) object //

:"The following example searches a string for the character "e //

;("!e/.exec("The best things in life are free/ //

JavaScript Operator Precedence //

Operator precedence describes the order in which operations are performed in an //
.arithmetic expression

Multiplication (*) and division (/) have higher precedence than addition (+) and subtraction //
.(-)

:As in traditional mathematics, multiplication is done first //

;let x = 100 + 50 * 3 //

:When using parentheses, operations inside the parentheses are computed first //

;let x = (100 + 50) * 3 //

:Operations with the same precedence (like * and /) are computed from left to right //

;let x = 100 / 50 * 3 //

Operator Precedence Values //
Expressions in parentheses are computed before the rest of the expression //
Function are executed before the result is used in the rest of the expression //
Example Description Operator Val //
(100 + 50) * 3 Expression Grouping ( ) 18 //
person.name Member Of . 17 //
["person["name Member Of [] 17 //
x ?. y Optional Chaining ES2020 .? 17 //
()myFunction Function Call () 17 //
("new Date("June 5,2022 New with Arguments new 17 //
()new Date New without Arguments new 16 //
Increment Operators //
Postfix increments are executed before prefix increments //
++i Postfix Increment ++ 15 //
--i Postfix Decrement -- 15 //
++i Prefix Increment ++ 14 //
--i Prefix Decrement -- 14 //
NOT Operators //
(!(x==y Logical NOT ! 14 //
~x Bitwise NOT ~ 14 //
Unary Operators //
+x Unary Plus + 14 //
-x Unary Minus - 14 //
typeof x Data Type typeof 14 //
(void(0 Evaluate Void void 14 //
delete myCar.color Property Delete delete 14 //

Arithmetic Operators //
Exponentiations are executed before multiplications //
Multiplications and divisions are executed before additions and subtractions //
10 ** 2 Exponentiation ES2016 ** 13 //
10 * 5 Multiplication * 12 //
10 / 5 Division / 12 //
10 % 5 Division Remainder % 12 //
10 + 5 Addition + 11 //
10 - 5 Subtraction - 11 //
""John" + "Doe Concatenation + 11 //
Shift Operators //
x << 2 Shift Left >> 10 //
x >> 2 Shift Right (signed) << 10 //
x >>> 2 Shift Right (unsigned) <<< 10 //
Relational Operators //
"PI" in Math Property in Object in 9 //
x instanceof Array Instance of Object instanceof 9 //
Comparison Operators //
x < y Less than > 9 //
x <= y Less than or equal => 9 //
x > y Greater than < 9 //
x >= Array Greater than or equal =< 9 //
x == y Equal == 8 //
x === y Strict equal === 8 //
x != y Unequal =! 8 //
x !== y Strict unequal ==! 8 //
Bitwise Operators //
x & y Bitwise AND & 7 //
x ^ y Bitwise XOR ^ 6 //
x | y Bitwise OR | 5 //
Logical Operators //
x && y Logical AND && 4 //
x || y Logical OR || 3 //
x ?? y Nullish Coalescing ES2020 ?? 3 //
Conditional (ternary) Operator //
"? "yes" : "no Condition : ? 2 //
Assignment Operators //
Assignments are executed after other operations //
x = y Simple Assignment = 2 //
x: 5 Colon Assignment : 2 //
x += y Addition Assignment =+ 2 //
x -= y Subtraction Assignment =- 2 //
x *= y Multiplication Assignment =* 2 //
x **= y Exponentiation Assignment =** 2 //
x /= y Division Assignment =/ 2 //
x %= y Remainder Assignment =% 2 //
x <<= y Left Shift Assignment =>> 2 //
x >>= y Right Shift Assignment =<< 2 //

x >>>= y Unsigned Right Shift =<<< 2 //
x &= y Bitwise AND Assignment =& 2 //
x |= y Bitwise OR Assignment =| 2 //
x ^= y Bitwise XOR Assignment =^ 2 //
x &= y Logical AND Assignment =&& 2 //
x ||= y Logical OR Assignment =|| 2 //
x => y Arrow <= 2 //
yield x Pause / Resume yield 2 //
yield* x Delegate yield* 2 //
... xSpread ... 2 //
x , y Comma , 1 //

JavaScript Errors //

Throw, and Try...Catch...Finally //
.(The try statement defines a code block to run (to try //

.The catch statement defines a code block to handle any error //

.The finally statement defines a code block to run regardless of the result //

.The throw statement defines a custom error //

!Errors Will Happen //
.When executing JavaScript code, different errors can occur //

Errors can be coding errors made by the programmer, errors due to wrong input, and other //
.unforeseeable things

 */}

<script>
} try
;("!adddlert("Welcome guest
{
} (catch(err
;document.getElementById("demo").innerHTML = err.message
{
{/* <script/>

JavaScript try and catch //
The try statement allows you to define a block of code to be tested for errors while it is //
.being executed

The catch statement allows you to define a block of code to be executed, if an error occurs //
.in the try block

:The JavaScript statements try and catch come in pairs //

} try //
Block of code to try   //
{ //
} (catch(err //
Block of code to handle errors   //
{ //

The throw Statement //
.The throw statement allows you to create a custom error //

.(Technically you can throw an exception (throw an error //

:The exception can be a JavaScript String, a Number, a Boolean or an Object //

throw "Too big";   // throw a text //
throw 500;   // throw a number //

If you use throw together with try and catch, you can control program flow and generate //
.custom error messages

Input Validation Example //
.This example examines input. If the value is wrong, an exception (err) is thrown //

The exception (err) is caught by the catch statement and a custom error message is //
:displayed

<DOCTYPE html!> //
<html> //
<body> //

Please input a number between 5 and 10: //

<"input id="demo" type="text> //
<button type="button" onclick="myFunction()">Test Input</button> //
 //

<script> //
} ()function myFunction //
;("const message = document.getElementById("p01   //
;"" = message.innerHTML   //
;let x = document.getElementById("demo").value   //
} try   //
;"if(x.trim() == "") throw "empty   //
;"if(isNaN(x)) throw "not a number   //
;(x = Number(x   //
;"if(x < 5) throw "too low   //
;"if(x > 10) throw "too high   //

```
{   //
} (catch(err   //
;message.innerHTML = "Input is " + err     //
{   //
{ //
<script/> //

<body/> //
<html/> //
```

HTML Validation //
.The code above is just an example //

Modern browsers will often use a combination of JavaScript and built-in HTML validation, //
:using predefined validation rules defined in HTML attributes

```
<"input id="demo" type="number" min="5" max="10" step="1> //
```
.You can read more about forms validation in a later chapter of this tutorial //

The finally Statement //
:The finally statement lets you execute code, after try and catch, regardless of the result //

Syntax //

```
} try //
Block of code to try   //
{ //
} (catch(err //
Block of code to handle errors   //
{ //
} finally //
Block of code to be executed regardless of the try / catch result   //
{ //

} ()function myFunction //
;("const message = document.getElementById("p01   //
;"" = message.innerHTML   //
;let x = document.getElementById("demo").value   //
} try   //
;"if(x.trim() == "") throw "is empty     //
;"if(isNaN(x)) throw "is not a number     //
;(x = Number(x     //
;"if(x > 10) throw "is too high     //
;"if(x < 5) throw "is too low     //
{   //
} (catch(err   //
;"." + message.innerHTML = "Error: " + err     //
{   //
} finally   //
```

;"" = document.getElementById("demo").value   //
{   //
{ //

The Error Object //
.JavaScript has a built in error object that provides error information when an error occurs //

.The error object provides two useful properties: name and message //

Error Object Properties //
Description   Property //
Sets or returns an error name   name //
(Sets or returns an error message (a string   message //

Error Name Values //
:Six different values can be returned by the error name property //

Description Error Name //
An error has occurred in the eval() function   EvalError //
A number "out of range" has occurred RangeError //
An illegal reference has occurred   ReferenceError //
A syntax error has occurred SyntaxError //
A type error has occurred   TypeError //
An error in encodeURI() has occurred   URIError //
.The six different values are described below //

Eval Error //
.An EvalError indicates an error in the eval() function //

.Newer versions of JavaScript do not throw EvalError. Use SyntaxError instead //

Range Error //
.A RangeError is thrown if you use a number that is outside the range of legal values //

.For example: You cannot set the number of significant digits of a number to 500 //

;let num = 1 //
} try //
num.toPrecision(500);   // A number cannot have 500 significant digits   //
{ //
} (catch(err //
;document.getElementById("demo").innerHTML = err.name   //
{ //

Reference Error //
:A ReferenceError is thrown if you use (reference) a variable that has not been declared //

;let x = 5 //

```
} try //
(x = y + 1;   // y cannot be used (referenced   //
{ //
} (catch(err //
;document.getElementById("demo").innerHTML = err.name   //
{ //
```

Syntax Error //
.A SyntaxError is thrown if you try to evaluate code with a syntax error //

```
} try //
eval("alert('Hello)");   // Missing ' will produce an error   //
{ //
} (catch(err //
;document.getElementById("demo").innerHTML = err.name   //
{ //
```

Type Error //
:A TypeError is thrown if you use a value that is outside the range of expected types //

```
;let num = 1 //
} try //
num.toUpperCase();   // You cannot convert a number to upper case   //
{ //
} (catch(err //
;document.getElementById("demo").innerHTML = err.name   //
{ //
```

URI (Uniform Resource Identifier) Error //
:A URIError is thrown if you use illegal characters in a URI function //

```
} try //
decodeURI("%%%");   // You cannot URI decode percent signs   //
{ //
} (catch(err //
;document.getElementById("demo").innerHTML = err.name   //
{ //
```

JavaScript Scope //

.Scope determines the accessibility (visibility) of variables //

:JavaScript has 3 types of scope //

Block scope //
Function scope //
Global scope //
Block Scope //

.Before ES6 (2015), JavaScript had only Global Scope and Function Scope //

.ES6 introduced two important new JavaScript keywords: let and const //

.These two keywords provide Block Scope in JavaScript //

:Variables declared inside a { } block cannot be accessed from outside the block //

} //
;let x = 2   //
{   //
x can NOT be used here //   //

.Variables declared with the var keyword can NOT have block scope //

.Variables declared inside a { } block can be accessed from outside the block //

} //
;var x = 2   //
{ //
x CAN be used here // //

Local Scope //
.Variables declared within a JavaScript function, become LOCAL to the function //

code here can NOT use carName // //

} ()function myFunction //
;"let carName = "Volvo   //
code here CAN use carName //   //
{ //

code here can NOT use carName // //

Function Scope //
.JavaScript has function scope: Each function creates a new scope //

.Variables defined inside a function are not accessible (visible) from outside the function //

.Variables declared with var, let and const are quite similar when declared inside a function //

:They all have Function Scope //

} ()function myFunction //
var carName = "Volvo";   // Function Scope   //
{ //

} ()function myFunction //

let carName = "Volvo";   // Function Scope   //
{   //

} ()function myFunction   //
const carName = "Volvo";   // Function Scope   //
{   //

Global JavaScript Variables //
.A variable declared outside a function, becomes GLOBAL //

;"let carName = "Volvo //
code here can use carName //

} ()function myFunction //
code here can also use carName //
{ //

Global Scope //
.Variables declared Globally (outside any function) have Global Scope //

.Global variables can be accessed from anywhere in a JavaScript program //

.Variables declared with var, let and const are quite similar when declared outside a block //

:They all have Global Scope //

var x = 2;   // Global scope //
let x = 2;   // Global scope //
const x = 2;   // Global scope //

JavaScript Variables //
.In JavaScript, objects and functions are also variables //

Automatically Global //
If you assign a value to a variable that has not been declared, it will automatically become //
.a GLOBAL variable

This code example will declare a global variable carName, even if the value is assigned //
.inside a function

;()myFunction //

code here can use carName //

} ()function myFunction //
;"carName = "Volvo   //
{ //

Strict Mode //
."All modern browsers support running JavaScript in "Strict Mode //

.You will learn more about how to use strict mode in a later chapter of this tutorial //

.In "Strict Mode", undeclared variables are not automatically global //

Global Variables in HTML //
.With JavaScript, the global scope is the JavaScript environment //

.In HTML, the global scope is the window object //

:Global variables defined with the var keyword belong to the window object //

;"var carName = "Volvo //
code here can use window.carName //

:Global variables defined with the let keyword do not belong to the window object //

;"let carName = "Volvo //
code here can not use window.carName //

Warning //
.Do NOT create global variables unless you intend to //

.(Your global variables (or functions) can overwrite window variables (or functions //
Any function, including the window object, can overwrite your global variables and //
.functions

The Lifetime of JavaScript Variables //
.The lifetime of a JavaScript variable starts when it is declared //

.Function (local) variables are deleted when the function is completed //

In a web browser, global variables are deleted when you close the browser window (or //
.(tab

Function Arguments //
.Function arguments (parameters) work as local variables inside functions //

JavaScript Hoisting //

.Hoisting is JavaScript's default behavior of moving declarations to the top //

JavaScript Declarations are Hoisted //
.In JavaScript, a variable can be declared after it has been used //

.In other words; a variable can be used before it has been declared //

:Example 1 gives the same result as Example 2 //

x = 5; // Assign 5 to x //

elem = document.getElementById("demo"); // Find an element //
elem.innerHTML = x;   // Display x in the element //

var x; // Declare x //

var x; // Declare x //
x = 5; // Assign 5 to x //

elem = document.getElementById("demo"); // Find an element //
elem.innerHTML = x;   // Display x in the element //

The let and const Keywords //
.Variables defined with let and const are hoisted to the top of the block, but not initialized //

Meaning: The block of code is aware of the variable, but it cannot be used until it has been //
.declared

.Using a let variable before it is declared will result in a ReferenceError //

:The variable is in a "temporal dead zone" from the start of the block until it is declared //

:This will result in a ReferenceError //
;"carName = "Volvo //
;let carName //

Using a const variable before it is declared, is a syntax error, so the code will simply not //
.run

.This code will not run //

;"carName = "Volvo //
;const carName //

JavaScript Initializations are Not Hoisted //
.JavaScript only hoists declarations, not initializations //

:Example 1 does not give the same result as Example 2 //

var x = 5; // Initialize x //
var y = 7; // Initialize y //

elem = document.getElementById("demo"); // Find an element //
elem.innerHTML = x + " " + y;   // Display x and y //

var x = 5; // Initialize x //

elem = document.getElementById("demo"); // Find an element //
elem.innerHTML = x + " " + y;   // Display x and y //

var y = 7; // Initialize y //

var x = 5; // Initialize x //
var y;   // Declare y //

elem = document.getElementById("demo"); // Find an element //
elem.innerHTML = x + " " + y;   // Display x and y //

y = 7;   // Assign 7 to y //

JavaScript Use Strict //

."use strict"; Defines that JavaScript code should be executed in "strict mode" //

Declaring Strict Mode //
.Strict mode is declared by adding "use strict"; to the beginning of a script or a function //

Declared at the beginning of a script, it has global scope (all code in the script will execute //
:(in strict mode

;"use strict" //
x = 3.14;   // This will cause an error because x is not declared //

;"use strict" //
;()myFunction //

} ()function myFunction //
y = 3.14;   // This will also cause an error because y is not declared   //
{ //

Declared inside a function, it has local scope (only the code inside the function is in strict //
:(mode

.x = 3.14;   // This will not cause an error //
;()myFunction //

} ()function myFunction //
;"use strict"   //
y = 3.14;   // This will cause an error   //
{ //

Not Allowed in Strict Mode //

:Using a variable, without declaring it, is not allowed //

;"use strict" //
x = 3.14;   // This will cause an erro //

:Using an object, without declaring it, is not allowed //

;"use strict" //
x = {p1:10, p2:20};   // This will cause an error //

.Deleting a variable (or object) is not allowed //

;"use strict" //
;let x = 3.14 //
delete x;   // This will cause an error //

.Deleting a function is not allowed //

;"use strict" //
;{} (function x(p1, p2 //
delete x;   // This will cause an error //

:Duplicating a parameter name is not allowed //

;"use strict" //
function x(p1, p1) {};   // This will cause an error //

:Octal numeric literals are not allowed //

;"use strict" //
let x = 010;   // This will cause an error //

:Octal escape characters are not allowed //

;"use strict" //
let x = "\010";   // This will cause an error //

:Writing to a read-only property is not allowed //

;"use strict" //
;{} = const obj //
;({Object.defineProperty(obj, "x", {value:0, writable:false //

obj.x = 3.14;   // This will cause an error //

:Writing to a get-only property is not allowed //

;"use strict" //

```
;{ {const obj = {get x() {return 0 //

obj.x = 3.14;            // This will cause an error //

:Deleting an undeletable property is not allowed //

;"use strict" //
delete Object.prototype; // This will cause an error //

:The word eval cannot be used as a variable //

;"use strict" //
let eval = 3.14;         // This will cause an error //

:The word arguments cannot be used as a variable //

;"use strict" //
let arguments = 3.14;    // This will cause an error //

:The with statement is not allowed //

;"use strict" //
with (Math){x = cos(2)}; // This will cause an error //

For security reasons, eval() is not allowed to create variables in the scope from which it //
.was called

:In strict mode, a variable can not be used before it is declared //

;"use strict" //
;("eval ("x = 2 //
alert (x);      // This will cause an error //

:In strict mode, eval() can not declare a variable using the var keyword //

;"use strict" //
;("eval ("var x = 2 //
alert (x);    // This will cause an error //

:eval() can not declare a variable using the let keyword //

;("eval ("let x = 2 //
alert (x);        // This will cause an error //

.The this keyword in functions behaves differently in strict mode //

.The this keyword refers to the object that called the function //
```

If the object is not specified, functions in strict mode will return undefined and functions in //
:(normal mode will return the global object (window

;"use strict" //
} ()function myFunction //
"alert(this); // will alert "undefined //
{ //

;()myFunction //

!Future Proof //
Keywords reserved for future JavaScript versions can NOT be used as variable names in //
.strict mode

:These are //

implements //
interface //
let //
package //
private //
protected //
public //
static //
yield //
;"use strict" //
let public = 1500; // This will cause an error //

!Watch Out //
.The "use strict" directive is only recognized at the beginning of a script or a function //

The JavaScript this Keyword //

} = const person //
,"firstName: "John //
,"lastName : "Doe //
,id : 5566 //
} ()fullName : function //
;return this.firstName + " " + this.lastName //
{ //
;{ //

?What is this //
.In JavaScript, the this keyword refers to an object //

.(Which object depends on how this is being invoked (used or called //

:The this keyword refers to different objects depending on how it is used //

.In an object method, this refers to the object //
.Alone, this refers to the global object //
.In a function, this refers to the global object //
.In a function, in strict mode, this is undefined //
.In an event, this refers to the element that received the event //
.Methods like call(), apply(), and bind() can refer this to any object //

Note //
.this is not a variable. It is a keyword. You cannot change the value of this //
this in a Method //
.When used in an object method, this refers to the object //

.In the example on top of this page, this refers to the person object //

.Because the fullName method is a method of the person object //

} ()fullName : function //
;return this.firstName + " " + this.lastName //
{ //

this Alone //
.When used alone, this refers to the global object //

.Because this is running in the global scope //

:[In a browser window the global object is [object Window //

;let x = this //

:In strict mode, when used alone, this also refers to the global object //

;"use strict" //
;let x = this //

(this in a Function (Default //
.In a function, the global object is the default binding for this //

:[In a browser window the global object is [object Window //

} ()function myFunction //
;return this //
{ //

(this in a Function (Strict //
.JavaScript strict mode does not allow default binding //

.So, when used in a function, in strict mode, this is undefined //

;"use strict" //
} ()function myFunction //
;return this //
{ //

this in Event Handlers //
:In HTML event handlers, this refers to the HTML element that received the event //

<'"button onclick="this.style.display='none> //
!Click to Remove Me //
<button/> //

Object Method Binding //
:In these examples, this is the person object //

} = const person //
,"firstName : "John //
,"lastName : "Doe //
,id : 5566 //
} ()myFunction : function //
;return this //
{ //
;{ //

} = const person //
,"firstName: "John //
,"lastName : "Doe //
,id : 5566 //
} ()fullName : function //
;return this.firstName + " " + this.lastName //
{ //
;{ //

.(i.e. this.firstName is the firstName property of this (the person object //

Explicit Function Binding //
.The call() and apply() methods are predefined JavaScript methods //

.They can both be used to call an object method with another object as argument //

The example below calls person1.fullName with person2 as an argument, this refers to //
:person2, even if fullName is a method of person1

} = const person1 //
} ()fullName: function //
;return this.firstName + " " + this.lastName //
{ //

{ //

} = const person2 //
,"firstName:"John //
,"lastName: "Doe //
{ //

:"Return "John Doe // //
;(person1.fullName.call(person2 //

Function Borrowing //
.With the bind() method, an object can borrow a method from another object //

.(This example creates 2 objects (person and member //

:The member object borrows the fullname method from the person object //

} = const person //
,"firstName:"John //
,"lastName: "Doe //
} () fullName: function //
;return this.firstName + " " + this.lastName //
{ //
{ //

} = const member //
,"firstName:"Hege //
,"lastName: "Nilsen //
{ //

;(let fullName = person.fullName.bind(member //

This Precedence //
.To determine which object this refers to; use the following precedence of order //

Object Precedence //
()bind 1 //
()apply() and call 2 //
Object method 3 //
Global scope 4 //
?()Is this in a function being called using bind //

?()Is this in a function being called using apply //

?()Is this in a function being called using call //

?(Is this in an object function (method //

.Is this in a function in the global scope //

JavaScript Arrow Function //

.Arrow functions were introduced in ES6 //

:Arrow functions allow us to write shorter function syntax //

```
;let myFunction = (a, b) => a * b //
```

:Before Arrow //

```
} ()hello = function //
;"!return "Hello World   //
{ //
```

:With Arrow Function //

```
} <= () = hello //
;"!return "Hello World   //
{ //
```

It gets shorter! If the function has only one statement, and the statement returns a value, //
:you can remove the brackets and the return keyword

:Arrow Functions Return Value by Default //

```
;"!hello = () => "Hello World //
```

:If you have parameters, you pass them inside the parentheses //

:Arrow Function With Parameters //

```
;hello = (val) => "Hello " + val //
```

:In fact, if you have only one parameter, you can skip the parentheses as well //

:Arrow Function Without Parentheses //

```
;hello = val => "Hello " + val //
```

:With a regular function this represents the object that calls the function //

:Regular Function //

```
} ()hello = function //
;document.getElementById("demo").innerHTML += this   //
{ //
```

:The window object calls the function //

```
;(window.addEventListener("load", hello //
```

:A button object calls the function //

```
;(document.getElementById("btn").addEventListener("click", hello //
```

:With an arrow function this represents the owner of the function //

:Arrow Function //
} <= () = hello //
;document.getElementById("demo").innerHTML += this //
{ //

:The window object calls the function //
;(window.addEventListener("load", hello //

:A button object calls the function //
;(document.getElementById("btn").addEventListener("click", hello //

JavaScript Classes //

JavaScript Class Syntax //
.Use the keyword class to create a class //

:()Always add a method named constructor //

Syntax //
} class ClassName //
{ ... } ()constructor //
{ //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
{ //

."The example above creates a class named "Car //

."The class has two initial properties: "name" and "year //

Using a Class //
:When you have a class, you can use the class to create objects //

;(const myCar1 = new Car("Ford", 2014 //
;(const myCar2 = new Car("Audi", 2019 //

.The example above uses the Car class to create two Car objects //

The Constructor Method //
:The constructor method is a special method //

"It has to have the exact name "constructor //
It is executed automatically when a new object is created //
It is used to initialize object properties //
If you do not define a constructor method, JavaScript will add an empty constructor //
.method

Class Methods //
.Class methods are created with the same syntax as object methods //

.Use the keyword class to create a class //

.Always add a constructor() method //

.Then add any number of methods //

Syntax //
} class ClassName //
{ ... } ()constructor //
{ ... } ()method_1 //
{ ... } ()method_2 //
{ ... } ()method_3 //
{ //

:Create a Class method named "age", that returns the Car age //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
} ()age //
;()const date = new Date //
;return date.getFullYear() - this.year //
{ //
{ //

;(const myCar = new Car("Ford", 2014 //
= document.getElementById("demo").innerHTML //
;".My car is " + myCar.age() + " years old" //

:You can send parameters to Class methods //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
} (age(x //

```
;return x - this.year     //
{   //
{ //

;()const date = new Date //
;()let year = date.getFullYear //

;(const myCar = new Car("Ford", 2014 //
=document.getElementById("demo").innerHTML //
;".My car is " + myCar.age(year) + " years old" //
```

JavaScript Modules //

Modules //
.JavaScript modules allow you to break up your code into separate files //

.This makes it easier to maintain a code-base //

.Modules are imported from external files with the import statement //

.Modules also rely on type="module" in the <script> tag //

```
<"script type="module> */}
;"import message from "./message.js
{/* <script/>
```

Export //
.Modules with functions or variables can be stored in any external file //

.There are two types of exports: Named Exports and Default Exports //

Named Exports //
.Let us create a file named person.js, and fill it with the things we want to export //

.You can create named exports two ways. In-line individually, or all at once at the bottom //

:In-line individually //
person.js //

```
;"export const name = "Jesse //
;export const age = 40 //
```

:All at once at the bottom //
person.js //

```
;"const name = "Jesse //
;const age = 40 //

;{export {name, age //
```

Default Exports //
.Let us create another file, named message.js, and use it for demonstrating default export //

.You can only have one default export in a file //

message.js //

} <= () = const message //
;"const name = "Jesse //
;const age = 40 //
;'.return name + ' is ' + age + 'years old //
;{ //

;export default message //

Import //
You can import modules into a file in two ways, based on if they are named exports or //
.default exports

.Named exports are constructed using curly braces. Default exports are not //

Import from named exports //
:Import named exports from the file person.js //

;"import { name, age } from "./person.js //

Import from default exports //
:Import a default export from the file message.js //

;"import message from "./message.js //

Note //
.Modules only work with the HTTP(s) protocol //

.A web-page opened via the file:// protocol cannot use import / export //

JavaScript JSON //

.JSON is a format for storing and transporting data //

.JSON is often used when data is sent from a server to a web page //

?What is JSON //
JSON stands for JavaScript Object Notation //
JSON is a lightweight data interchange format //
* JSON is language independent //
JSON is "self-describing" and easy to understand //

The JSON syntax is derived from JavaScript object notation syntax, but the JSON format * //
is text only. Code for reading and generating JSON data can be written in any programming
.language

JSON Example //
:(This JSON syntax defines an employees object: an array of 3 employee records (objects //

JSON Example //
} //
]:"employees" //
,{"firstName":"John", "lastName":"Doe"} //
,{"firstName":"Anna", "lastName":"Smith"} //
{"firstName":"Peter", "lastName":"Jones"} //
[//
{ //
The JSON Format Evaluates to JavaScript Objects //
.The JSON format is syntactically identical to the code for creating JavaScript objects //

Because of this similarity, a JavaScript program can easily convert JSON data into native //
.JavaScript objects

JSON Syntax Rules //
Data is in name/value pairs //
Data is separated by commas //
Curly braces hold objects //
Square brackets hold arrays //

JSON Data - A Name and a Value //
.JSON data is written as name/value pairs, just like JavaScript object properties //

A name/value pair consists of a field name (in double quotes), followed by a colon, //
:followed by a value

"firstName":"John" //
.JSON names require double quotes. JavaScript names do not //

JSON Objects //
.JSON objects are written inside curly braces //

:Just like in JavaScript, objects can contain multiple name/value pairs //

{"firstName":"John", "lastName":"Doe"} //
JSON Arrays //
.JSON arrays are written inside square brackets //

:Just like in JavaScript, an array can contain objects //

]:"employees" //

,{"firstName":"John", "lastName":"Doe"}   //
,{"firstName":"Anna", "lastName":"Smith"}   //
{"firstName":"Peter", "lastName":"Jones"}   //
[//
.In the example above, the object "employees" is an array. It contains three objects //

.(Each object is a record of a person (with a first name and a last name //

Converting a JSON Text to a JavaScript Object //
A common use of JSON is to read data from a web server, and display the data in a web //
.page

.For simplicity, this can be demonstrated using a string as input //

:First, create a JavaScript string containing JSON syntax //

+ '] : "let text = '{ "employees //
+ ',{ "firstName":"John" , "lastName":"Doe" }' //
+ ',{ "firstName":"Anna" , "lastName":"Smith" }' //
;'{[{ "firstName":"Peter" , "lastName":"Jones" }' //
Then, use the JavaScript built-in function JSON.parse() to convert the string into a //
:JavaScript object

;(const obj = JSON.parse(text //
:Finally, use the new JavaScript object in your page //

Example //
 //

<script> //
= document.getElementById("demo").innerHTML //
;obj.employees[1].firstName + " " + obj.employees[1].lastName //
<script/> //

JavaScript Debugging //

The console.log() Method //
If your browser supports debugging, you can use console.log() to display JavaScript //
:values in the debugger window

<DOCTYPE html!> //
<html> //
<body> //

<h1>My First Web Page</h1> //

<script> //
;a = 5 //

;b = 6 //
;c = a + b //
;(console.log(c //
<script/> //

<body/> //
<html/> //

The debugger Keyword //
The debugger keyword stops the execution of JavaScript, and calls (if available) the //
.debugging function

.This has the same function as setting a breakpoint in the debugger //

.If no debugging is available, the debugger statement has no effect //

.With the debugger turned on, this code will stop executing before it executes the third line //

;let x = 15 * 5 //
;debugger //
;document.getElementById("demo").innerHTML = x //

Major Browsers' Debugging Tools //
Normally, you activate debugging in your browser with F12, and select "Console" in the //
.debugger menu

:Otherwise follow these steps //

Chrome //
.Open the browser //
."From the menu, select "More tools //
."From tools, choose "Developer tools //
.Finally, select Console //
Firefox //
.Open the browser //
."From the menu, select "Web Developer //
."Finally, select "Web Console //
Edge //
.Open the browser //
."From the menu, select "Developer Tools //
."Finally, select "Console //
Opera //
.Open the browser //
."From the menu, select "Developer //
."From "Developer", select "Developer tools //
."Finally, select "Console //
Safari //
.Go to Safari, Preferences, Advanced in the main menu //

."Check "Enable Show Develop menu in menu bar //
:When the new option "Develop" appears in the menu //
."Choose "Show Error Console //

JavaScript Style Guide //

.Always use the same coding conventions for all your JavaScript projects //

JavaScript Coding Conventions //
:Coding conventions are style guidelines for programming. They typically cover //

.Naming and declaration rules for variables and functions //
.Rules for the use of white space, indentation, and comments //
.Programming practices and principles //
:Coding conventions secure quality //

Improve code readability //
Make code maintenance easier //
Coding conventions can be documented rules for teams to follow, or just be your individual //
.coding practice

Variable Names //
.(At W3schools we use camelCase for identifier names (variables and functions //

.All names start with a letter //

."firstName = "John //
."lastName = "Doe //

;price = 19.90 //
;tax = 0.20 //

;(fullPrice = price + (price * tax //
Spaces Around Operators //
:Always put spaces around operators (= + - * /), and after commas //

:Examples //
;let x = y + z //
;["const myArray = ["Volvo", "Saab", "Fiat //

Code Indentation //
:Always use 2 spaces for indentation of code blocks //

:Functions //
} (function toCelsius(fahrenheit //
;(return (5 / 9) * (fahrenheit - 32 //

{ //

.Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently //

Statement Rules //
:General rules for simple statements //

.Always end a simple statement with a semicolon //
:Examples //
;["const cars = ["Volvo", "Saab", "Fiat //

} = const person //
,"firstName: "John //
,"lastName: "Doe //
,age: 50 //
"eyeColor: "blue //
;{ //
:General rules for complex (compound) statements //

.Put the opening bracket at the end of the first line //
.Use one space before the opening bracket //
.Put the closing bracket on a new line, without leading spaces //
.Do not end a complex statement with a semicolon //
:Functions //
} (function toCelsius(fahrenheit //
;(return (5 / 9) * (fahrenheit - 32 //
{ //
:Loops //
} (++for (let i = 0; i < 5; i //
;x += i //
{ //
:Conditionals //
} (if (time < 20 //
;"greeting = "Good day //
} else { //
;"greeting = "Good evening //
{ //
Object Rules //
:General rules for object definitions //

.Place the opening bracket on the same line as the object name //
.Use colon plus one space between each property and its value //
.Use quotes around string values, not around numeric values //
.Do not add a comma after the last property-value pair //
.Place the closing bracket on a new line, without leading spaces //
.Always end an object definition with a semicolon //
Example //
} = const person //

,"firstName: "John   //
,"lastName: "Doe   //
,age: 50   //
"eyeColor: "blue   //
;{ //
Short objects can be written compressed, on one line, using spaces only between //
:properties, like this

;{"const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //
Line Length < 80 //
.For readability, avoid lines longer than 80 characters //

If a JavaScript statement does not fit on one line, the best place to break it, is after an //
.operator or a comma

Example //
= document.getElementById("demo").innerHTML //
;".Hello Dolly" //

Naming Conventions //
:Always use the same naming convention for all your code. For example //

Variable and function names written as camelCase //
(Global variables written in UPPERCASE (We don't, but it's quite common //
Constants (like PI) written in UPPERCASE //
?Should you use hyp-hens, camelCase, or under_scores in variable names //

:This is a question programmers often discuss. The answer depends on who you ask //

:Hyphens in HTML and CSS //

.(HTML5 attributes can start with data- (data-quantity, data-price //

.(CSS uses hyphens in property-names (font-size //

Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript //
.names

:Underscores //

Many programmers prefer to use underscores (date_of_birth), especially in SQL //
.databases

.Underscores are often used in PHP documentation //

:PascalCase //

.PascalCase is often preferred by C programmers //

:camelCase //

.camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries //

Do not start names with a $ sign. It will put you in conflict with many JavaScript library //
.names

Loading JavaScript in HTML //
:(Use simple syntax for loading external scripts (the type attribute is not necessary //

<script src="myscript.js"></script> //
Accessing HTML Elements //
.A consequence of using "untidy" HTML styles, might result in JavaScript errors //

:These two JavaScript statements will produce different results //

("const obj = getElementById("Demo //

("const obj = getElementById("demo //
.If possible, use the same naming convention (as JavaScript) in HTML //

.Visit the HTML Style Guide //

File Extensions //
.(HTML files should have a .html extension (.htm is allowed //

.CSS files should have a .css extension //

.JavaScript files should have a .js extension //

Use Lower Case File Names //
:Most web servers (Apache, Unix) are case sensitive about file names //

.london.jpg cannot be accessed as London.jpg //

:Other web servers (Microsoft, IIS) are not case sensitive //

.london.jpg can be accessed as London.jpg or london.jpg //

.If you use a mix of upper and lower case, you have to be extremely consistent //

If you move from a case insensitive, to a case sensitive server, even small errors can //
.break your web site

.(To avoid these problems, always use lower case file names (if possible //

Performance //

Coding conventions are not used by computers. Most rules have little impact on the //
.execution of programs

.Indentation and extra spaces are not significant in small scripts //

For code in development, readability should be preferred. Larger production scripts should //
.be minimized

JavaScript Best Practices //

()Avoid global variables, avoid new, avoid ==, avoid eval //

Always Declare Local Variables //
.All variables used in a function should be declared as local variables //

Local variables must be declared with the var, the let, or the const keyword, otherwise they //
.will become global variables

Declarations on Top //
.It is a good coding practice to put all declarations at the top of each script or function //

:This will //

Give cleaner code //
Provide a single place to look for local variables //
Make it easier to avoid unwanted (implied) global variables //
Reduce the possibility of unwanted re-declarations //
Declare at the beginning // //
;let firstName, lastName, price, discount, fullPrice //

Use later // //
;"firstName = "John //
;"lastName = "Doe //

;price = 19.90 //
;discount = 0.10 //

;fullPrice = price - discount //
:This also goes for loop variables //

} (++for (let i = 0; i < 5; i //

Initialize Variables //
.It is a good coding practice to initialize variables when you declare them //

:This will //

Give cleaner code //

Provide a single place to initialize variables //
Avoid undefined values //
Declare and initiate at the beginning // //
;"" = let firstName //
;"" = let lastName //
;let price = 0 //
;let discount = 0 //
,let fullPrice = 0 //
;[] = const myArray //
;{} = const myObject //

Declare Objects with const //
:Declaring objects with const will prevent any accidental change of type //

Example //
;{"let car = {type:"Fiat", model:"500", color:"white //
car = "Fiat";   // Changes object to string //

;{"const car = {type:"Fiat", model:"500", color:"white //
car = "Fiat";   // Not possible //

Declare Arrays with const //
:Declaring arrays with const will prevent any accidential change of type //

Example //
;["let cars = ["Saab", "Volvo", "BMW //
cars = 3;   // Changes array to number //

;["const cars = ["Saab", "Volvo", "BMW //
cars = 3;   // Not possible //

()Don't Use new Object //
()Use "" instead of new String //
()Use 0 instead of new Number //
()Use false instead of new Boolean //
()Use {} instead of new Object //
()Use [] instead of new Array //
()Use /()/ instead of new RegExp //
()Use function (){} instead of new Function //

```
let x1 = "";            // new primitive string //
let x2 = 0;             // new primitive number //
let x3 = false;         // new primitive boolean //
const x4 = {};          // new object //
const x5 = [];          // new array object //
const x6 = /()/;        // new regexp object //
const x7 = function(){}; // new function object //
```

Beware of Automatic Type Conversions //
.JavaScript is loosely typed //

.A variable can contain all data types //

:A variable can change its data type //

```
let x = "Hello";     // typeof x is a string //
x = 5;               // changes typeof x to a number //
```

.(Beware that numbers can accidentally be converted to strings or NaN (Not a Number //

:When doing mathematical operations, JavaScript can convert numbers to strings //

```
let x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number //
let x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string //
let x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string //
let x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number //
let x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number //
let x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number //
let x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number //
```

Subtracting a string from a string, does not generate an error but returns NaN (Not a //
:(Number

```
Hello" - "Dolly"    // returns NaN" //
```

Use === Comparison //
.The == comparison operator always converts (to matching types) before comparison //

:The === operator forces comparison of values and type //

```
true //        ;"" == 0 //
true //       ;"1" == 1 //
true;      // true == 1 //

false //       ;"" === 0 //
false //      ;"1" === 1 //
true;     // false === 1 //
```

Use Parameter Defaults //
If a function is called with a missing argument, the value of the missing argument is set to //
.undefined

Undefined values can break your code. It is a good habit to assign default values to //
.arguments

```
} (function myFunction(x, y //
```

} (if (y === undefined   //
;y = 0   //
{   //
{ //

{ /*function (a=1, b=1) { /*function code //

End Your Switches with Defaults //
.Always end your switch statements with a default. Even if you think there is no need for it //

} (()switch (new Date().getDay //
:case 0   //
;"day = "Sunday   //
;break   //
:case 1   //
;"day = "Monday   //
;break   //
:case 2   //
;"day = "Tuesday   //
;break   //
:case 3   //
;"day = "Wednesday   //
;break   //
:case 4   //
;"day = "Thursday   //
;break   //
:case 5   //
;"day = "Friday   //
;break   //
:case 6   //
;"day = "Saturday   //
;break   //
:default   //
;"day = "Unknown   //
{ //

Avoid Number, String, and Boolean as Objects //
.Always treat numbers, strings, or booleans as primitive values. Not as objects //

Declaring these types as objects, slows down execution speed, and produces nasty side //
:effects

;"let x = "John //
;("let y = new String("John //
.x === y) // is false because x is a string and y is an object) //

:Or even worse //

;("let x = new String("John //
;("let y = new String("John //
.x == y) // is false because you cannot compare objects) //

()Avoid Using eval //
The eval() function is used to run text as code. In almost all cases, it should not be //
.necessary to use it

.Because it allows arbitrary code to be run, it also represents a security problem //

JavaScript Common Mistakes //

.This chapter points out some common JavaScript mistakes //

Accidentally Using the Assignment Operator //
JavaScript programs may generate unexpected results if a programmer accidentally uses //
.an assignment operator (=), instead of a comparison operator (==) in an if statement

:This if statement returns false (as expected) because x is not equal to 10 //

;let x = 0 //
(if (x == 10 //

:This if statement returns true (maybe not as expected), because 10 is true //

;let x = 0 //
(if (x = 10 //

:This if statement returns false (maybe not as expected), because 0 is false //

;let x = 0 //
(if (x = 0 //

Expecting Loose Comparison //
:In regular comparison, data type does not matter. This if statement returns true //

;let x = 10 //
;"let y = "10 //
(if (x == y //

:In strict comparison, data type does matter. This if statement returns false //

;let x = 10 //
;"let y = "10 //
(if (x === y //

:It is a common mistake to forget that switch statements use strict comparison //

:This case switch will display an alert //

;let x = 10 //
} (switch(x //
;("case 10: alert("Hello //
{ //

:This case switch will not display an alert //

;let x = 10 //
} (switch(x //
;("case "10": alert("Hello //
{ //

Confusing Addition & Concatenation //
.Addition is about adding numbers //

.Concatenation is about adding strings //

.In JavaScript both operations use the same + operator //

Because of this, adding a number as a number will produce a different result from adding a //
:number as a string

;let x = 10 //
x = 10 + 5;   // Now x is 15 //

;let y = 10 //
"y += "5";   // Now y is "105 //

:When adding two variables, it can be difficult to anticipate the result //

;let x = 10 //
;let y = 5 //
let z = x + y;   // Now z is 15 //

;let x = 10 //
;"let y = "5 //
"let z = x + y;   // Now z is "105 //

Misunderstanding Floats //
.(All numbers in JavaScript are stored as 64-bits Floating point numbers (Floats //

All programming languages, including JavaScript, have difficulties with precise floating //
:point values

;let x = 0.1 //
;let y = 0.2 //

let z = x + y   // the result in z will not be 0.3 //

:To solve the problem above, it helps to multiply and divide //

let z = (x * 10 + y * 10) / 10;   // z will be 0.3 //

Breaking a JavaScript String //
:JavaScript will allow you to break a statement into two lines //

= let x //
;"!Hello World" //

:But, breaking a statement in the middle of a string will not work //

let x = "Hello //
;"!World //

:You must use a "backslash" if you must break a statement in a string //

\ let x = "Hello //
;"!World //

Misplacing Semicolon //
Because of a misplaced semicolon, this code block will execute regardless of the value of //
:x

;(if (x == 19 //
} //
code block //   //
{ //

Breaking a Return Statement //
.It is a default JavaScript behavior to close a statement automatically at the end of a line //

:Because of this, these two examples will return the same result //

} (function myFunction(a //
let power = 10   //
return a * power   //
{ //

} (function myFunction(a //
;let power = 10   //
;return a * power   //
{   //

.JavaScript will also allow you to break a statement into two lines //

:Because of this, example 3 will also return the same result //

```
} (function myFunction(a //
let   //
;power = 10   //
;return a * power   //
{ //
```

:But, what will happen if you break the return statement in two lines like this //

```
} (function myFunction(a //
let   //
;power = 10   //
return   //
;a * power   //
{ //
```

!The function will return undefined //

:Why? Because JavaScript thought you meant //

```
} (function myFunction(a //
let   //
;power = 10   //
;return   //
;a * power   //
{ //
```

Explanation //
:If a statement is incomplete like //

let //
:JavaScript will try to complete the statement by reading the next line //

;power = 10 //
:But since this statement is complete //

return //
:JavaScript will automatically close it like this //

;return //
This happens because closing (ending) statements with semicolon is optional in //
.JavaScript

JavaScript will close the return statement at the end of the line, because it is a complete //
.statement

.Never break a return statement //

Accessing Arrays with Named Indexes //
.Many programming languages support arrays with named indexes //

.(Arrays with named indexes are called associative arrays (or hashes //

.JavaScript does not support arrays with named indexes //

:In JavaScript, arrays use numbered indexes //

```
;[] = const person //
;"person[0] = "John //
;"person[1] = "Doe //
;person[2] = 46 //
person.length;       // person.length will return 3 //
"person[0];          // person[0] will return "John //
```

.In JavaScript, objects use named indexes //

If you use a named index, when accessing an array, JavaScript will redefine the array to a //
.standard object

After the automatic redefinition, array methods and properties will produce undefined or //
:incorrect results

```
;[] = const person //
;"person["firstName"] = "John //
;"person["lastName"] = "Doe //
;person["age"] = 46 //
person.length;      // person.length will return 0 //
person[0];          // person[0] will return undefined //
```

Ending Definitions with a Comma //
.Trailing commas in object and array definition are legal in ECMAScript 5 //

:Object Example //

```
{,person = {firstName:"John", lastName:"Doe", age:46 //
```

:Array Example //

```
;[,points = [40, 100, 1, 5, 25, 10 //
```

!! WARNING //

.Internet Explorer 8 will crash //

.JSON does not allow trailing commas //

:JSON //

```
{person = {"firstName":"John", "lastName":"Doe", "age":46 //
```

:JSON //
;[points = [40, 100, 1, 5, 25, 10 //
Undefined is Not Null //
.JavaScript objects, variables, properties, and methods can be undefined //

.In addition, empty JavaScript objects can have the value null //

.This can make it a little bit difficult to test if an object is empty //

:You can test if an object exists by testing if the type is undefined //

:Example //
("if (typeof myObj === "undefined //

But you cannot test if an object is null, because this will throw an error if the object is //
:undefined

:Incorrect //
(if (myObj === null //
.To solve this problem, you must test if an object is not null, and not undefined //

:But this can still throw an error //

:Incorrect //
("if (myObj !== null && typeof myObj !== "undefined //
:Because of this, you must test for not undefined before you can test for not null //

:Correct //
(if (typeof myObj !== "undefined" && myObj !== null //

JavaScript Performance //

.How to speed up your JavaScript code //

Reduce Activity in Loops //
.Loops are often used in programming //

Each statement in a loop, including the for statement, is executed for each iteration of the //
.loop

Statements or assignments that can be placed outside the loop will make the loop run //
.faster

:Bad //
} (++for (let i = 0; i < arr.length; i //

:Better Code //
;let l = arr.length //

} (++for (let i = 0; i < l; i //
.The bad code accesses the length property of an array each time the loop is iterated //

The better code accesses the length property outside the loop and makes the loop run //
.faster

Reduce DOM Access //
.Accessing the HTML DOM is very slow, compared to other JavaScript statements //

If you expect to access a DOM element several times, access it once, and use it as a local //
:variable

Example //
;("const obj = document.getElementById("demo //
;"obj.innerHTML = "Hello //

Reduce DOM Size //
.Keep the number of elements in the HTML DOM small //

This will always improve page loading, and speed up rendering (page display), especially //
.on smaller devices

Every attempt to search the DOM (like getElementsByTagName) will benefit from a smaller //
.DOM

Avoid Unnecessary Variables //
.Don't create new variables if you don't plan to save values //

:Often you can replace code like this //

;let fullName = firstName + " " + lastName //
;document.getElementById("demo").innerHTML = fullName //
:With this //

;document.getElementById("demo").innerHTML = firstName + " " + lastName //
Delay JavaScript Loading //
.Putting your scripts at the bottom of the page body lets the browser load the page first //

While a script is downloading, the browser will not start any other downloads. In addition all //
.parsing and rendering activity might be blocked

The HTTP specification defines that browsers should not download more than two //
.components in parallel

An alternative is to use defer="true" in the script tag. The defer attribute specifies that the //
script should be executed after the page has finished parsing, but it only works for external
.scripts

:If possible, you can add your script to the page by code, after the page has loaded //

Example //
<script> //
} ()window.onload = function //
;("const element = document.createElement("script //
;"element.src = "myScript.js //
;(document.body.appendChild(element //
;{ //
<script/> //
Avoid Using with //
Avoid using the with keyword. It has a negative effect on speed. It also clutters up //
.JavaScript scopes

.The with keyword is not allowed in strict mode //

JavaScript Reserved Words //

In JavaScript you cannot use these reserved words as variables, labels, or function //
:names
boolean await* arguments abstract //
catch case byte break //
continue const class* char //
do delete default debugger //
eval enum* else double //
final false extends* export* //
function for float finally //
*import implements if goto //
interface int instanceof in //
new native long let* //
protected private package null //
static short return public //
this synchronized switch super* //
true transient throws throw //
void var typeof try //
yield with while volatile //
.Words marked with* are new in ECMAScript 5 and 6 //

Removed Reserved Words //
:The following reserved words have been removed from the ECMAScript 5/6 standard //
char byte boolean abstract //
goto float final double //
short native long int //
volatile transient throws synchronized //

avaScript Objects, Properties, and Methods //
You should also avoid using the name of JavaScript built-in objects, properties, and //
:methods

function eval DateArray //
isNaN isFinite Infinity hasOwnProperty //
NaN Math length isPrototypeOf //
prototype Object Number name //
valueOf undefined toString String //

Java Reserved Words //
JavaScript is often used together with Java. You should avoid using some Java objects //
:and properties as JavaScript identifiers
javaClass JavaArray java getClass //
JavaPackage JavaObject //
Other Reserved Words //
.JavaScript can be used as the programming language in many applications //

:You should also avoid using the name of HTML and Window objects and properties //
anchors anchor all alert //
button blur assign area //
clientInformation clearTimeout clearInterval checkbox //
constructor confirm closed close //
defaultStatus decodeURIComponent decodeURI crypto //
embed elements element document //
escape encodeURIComponent encodeURI embeds //
form focus fileUpload event //
innerWidth innerHeight frame forms //
location link layers layer //
frames navigator navigate mimeTypes //
image history hidden frameRate //
opener open offscreenBuffering images //
packages outerWidth outerHeight option //
parseFloat parent pageYOffset pageXOffset //
plugin pkcs11 password parseInt //
reset radio propertyIsEnum prompt //
secure scroll screenY screenX //
setTimeout setInterval self select //
text taint submit status //
untaint unescape top textarea //
window //
HTML Event Handlers //
.In addition you should avoid using the name of all HTML event handlers //

onfocus onerror onclick onblur //
onmouseover onkeyup onkeypress onkeydown //
onsubmit onmousedown onmouseup onload //

JavaScript Objects //

.In JavaScript, objects are king. If you understand objects, you understand JavaScript //

.In JavaScript, almost "everything" is an object //

(Booleans can be objects (if defined with the new keyword //
(Numbers can be objects (if defined with the new keyword //
(Strings can be objects (if defined with the new keyword //
Dates are always objects //
Maths are always objects //
Regular expressions are always objects //
Arrays are always objects //
Functions are always objects //
Objects are always objects //
.All JavaScript values, except primitives, are objects //

JavaScript Primitives //
.A primitive value is a value that has no properties or methods //

is a primitive value 3.14 //

.A primitive data type is data that has a primitive value //

:JavaScript defines 7 types of primitive data types //

Examples //
string //
number //
boolean //
null //
undefined //
symbol //
bigint //
Immutable //
.(Primitive values are immutable (they are hardcoded and cannot be changed //

.if x = 3.14, you can change the value of x, but you cannot change the value of 3.14 //

Comment Type Value //
""Hello" is always "Hello string Hello"" //
3.14 is always 3.14 number 3.14 //
true is always true boolean true //
false is always false boolean false //
null is always null null (object) null //
undefined is always undefined undefined undefined //
Objects are Variables //
:JavaScript variables can contain single values //

;"let person = "John Doe //

.JavaScript variables can also contain many values //

.Objects are variables too. But objects can contain many values //

.(Object values are written as name : value pairs (name and value separated by a colon //

;{"let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //

A JavaScript object is a collection of named values //

.It is a common practice to declare objects with the const keyword //

;{"const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //

Creating a JavaScript Object //
.With JavaScript, you can define and create your own objects //

:There are different ways to create new objects //

.Create a single object, using an object literal //
.Create a single object, with the keyword new //
.Define an object constructor, and then create objects of the constructed type //
.()Create an object using Object.create //
Using an Object Literal //
.This is the easiest way to create a JavaScript Object //

.Using an object literal, you both define and create an object in one statement //

.{} An object literal is a list of name:value pairs (like age:50) inside curly braces //

:The following example creates a new JavaScript object with four properties //

;{"const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue //

:Spaces and line breaks are not important. An object definition can span multiple lines //

} = const person //
,"firstName: "John //
,"lastName: "Doe //
,age: 50 //
"eyeColor: "blue //
;{ //

:This example creates an empty JavaScript object, and then adds 4 properties //

;{} = const person //
;"person.firstName = "John //
;"person.lastName = "Doe //
;person.age = 50 //

;"person.eyeColor = "blue //

Using the JavaScript Keyword new //
The following example create a new JavaScript object using new Object(), and then adds 4 //
:properties

;()const person = new Object //
;"person.firstName = "John //
;"person.lastName = "Doe //
;person.age = 50 //
;"person.eyeColor = "blue //

.The examples above do exactly the same //

.()But there is no need to use new Object //

.For readability, simplicity and execution speed, use the object literal method //

JavaScript Objects are Mutable //
.Objects are mutable: They are addressed by reference, not by value //

:If person is an object, the following statement will not create a copy of person //

.const x = person; // Will not create a copy of person //
.The object x is not a copy of person. It is person. Both x and person are the same object //

.Any changes to x will also change person, because x and person are the same object //

} = const person //
,"firstName:"John   //
,"lastName:"Doe   //
"age:50, eyeColor:"blue   //
{ //

;const x = person //
x.age = 10;   // Will change both x.age and person.age //

JavaScript Object Properties //

.Properties are the most important part of any JavaScript object //

JavaScript Properties //
.Properties are the values associated with a JavaScript object //

.A JavaScript object is a collection of unordered properties //

.Properties can usually be changed, added, and deleted, but some are read only //

Accessing JavaScript Properties //
:The syntax for accessing the property of an object is //

objectName.property   // person.age //
or //

["objectName["property"]   // person["age //
or //

[objectName[expression]   // x = "age"; person[x //
.The expression must evaluate to a property name //

;".person.firstname + " is " + person.age + " years old //

;".person["firstname"] + " is " + person["age"] + " years old //

JavaScript for...in Loop //
.The JavaScript for...in statement loops through the properties of an object //

Syntax //
} (for (let variable in object //
code to be executed //
{ //
.The block of code inside of the for...in loop will be executed once for each property //

:Looping through the properties of an object //

} = const person //
,"fname:" John   //
,"lname:" Doe   //
age: 25   //
;{ //

} (for (let x in person //
;[txt += person[x   //
{ //

Adding New Properties //
.You can add new properties to an existing object by simply giving it a value //

:Assume that the person object already exists - you can then give it new properties //

;"person.nationality = "English //

Deleting Properties //
:The delete keyword deletes a property from an object //

} = const person //

,"firstName: "John   //
,"lastName: "Doe   //
,age: 50   //
"eyeColor: "blue   //
;{ //

;delete person.age //

;["or delete person["age //

} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
,age: 50   //
"eyeColor: "blue   //
;{ //

;["delete person["age //

.The delete keyword deletes both the value of the property and the property itself //

.After deletion, the property cannot be used before it is added back again //

The delete operator is designed to be used on object properties. It has no effect on //
.variables or functions

The delete operator should not be used on predefined JavaScript object properties. It can //
.crash your application

Nested Objects //
:Values in an object can be another object //

} = myObj //
,"name:"John   //
,age:30   //
} :cars   //
,"car1:"Ford   //
,"car2:"BMW   //
"car3:"Fiat   //
{   //
{ //
:You can access nested objects using the dot notation or the bracket notation //

;myObj.cars.car2 //

:or //

;["myObj.cars["car2 //

:or //

;["myObj["cars"]["car2 //

:or //

;"let p1 = "cars //
;"let p2 = "car2 //
;[myObj[p1][p2 //

Nested Arrays and Objects //
:Values in objects can be arrays, and values in arrays can be objects //

} = const myObj //
,"name: "John //
,age: 30 //
] :cars //
,{["name:"Ford", models:["Fiesta", "Focus", "Mustang} //
,{["name:"BMW", models:["320", "X3", "X5} //
{["name:"Fiat", models:["500", "Panda} //
[//
{ //

:To access arrays inside arrays, use a for-in loop for each array //

} (for (let i in myObj.cars //
;"<x += "<h1>" + myObj.cars[i].name + "</h1 //
} (for (let j in myObj.cars[i].models //
;[x += myObj.cars[i].models[j //
{ //
{ //

JavaScript Object Methods //

} = const person //
,"firstName: "John //
,"lastName: "Doe //
,id: 5566 //
} ()fullName: function //
;return this.firstName + " " + this.lastName //
{ //
;{ //

?What is this //
.In JavaScript, the this keyword refers to an object //

.(Which object depends on how this is being invoked (used or called //

:The this keyword refers to different objects depending on how it is used //

.In an object method, this refers to the object //
.Alone, this refers to the global object //
.In a function, this refers to the global object //
.In a function, in strict mode, this is undefined //
.In an event, this refers to the element that received the event //
.Methods like call(), apply(), and bind() can refer this to any object //
Note //
.this is not a variable. It is a keyword. You cannot change the value of this //
:See Also //
The JavaScript this Tutorial //

JavaScript Methods //
.JavaScript methods are actions that can be performed on objects //

.A JavaScript method is a property containing a function definition //

Value Property //
John firstName //
Doe lastName //
50 age //
blue eyeColor //
{;function() {return this.firstName + " " + this.lastName fullName //
.Methods are functions stored as object properties //

Accessing Object Methods //
:You access an object method with the following syntax //

()objectName.methodName //
You will typically describe fullName() as a method of the person object, and fullName as a //
.property

.() The fullName property will execute (as a function) when it is invoked with //

:This example accesses the fullName() method of a person object //

;()name = person.fullName //

:If you access the fullName property, without (), it will return the function definition //

;name = person.fullName //

Adding a Method to an Object //
:Adding a new method to an object is easy //

} () person.name = function //

;return this.firstName + " " + this.lastName //
;{ //

Using Built-In Methods //
This example uses the toUpperCase() method of the String object, to convert a text to //
:uppercase

;"!let message = "Hello world //
;()let x = message.toUpperCase //
:The value of x, after execution of the code above will be //

!HELLO WORLD //

} () person.name = function //
;()return (this.firstName + " " + this.lastName).toUpperCase //
;{ //

JavaScript Display Objects //

?How to Display JavaScript Objects //
.[Displaying a JavaScript object will output [object Object //

} = const person //
,"name: "John //
,age: 30 //
"city: "New York //
;{ //

;document.getElementById("demo").innerHTML = person //

:Some common solutions to display JavaScript objects are //

Displaying the Object Properties by name //
Displaying the Object Properties in a Loop //
()Displaying the Object using Object.values //
()Displaying the Object using JSON.stringify //
Displaying Object Properties //
:The properties of an object can be displayed as a string //

} = const person //
,"name: "John //
,age: 30 //
"city: "New York //
;{ //

= document.getElementById("demo").innerHTML //
;person.name + "," + person.age + "," + person.city //

Displaying the Object in a Loop //
:The properties of an object can be collected in a loop //

} = const person //
,"name: "John //
,age: 30 //
"city: "New York //
;{ //

;"" = let txt //
} (for (let x in person //
;" " + [txt += person[x //
;{ //

;document.getElementById("demo").innerHTML = txt //

.You must use person[x] in the loop //

.(person.x will not work (Because x is a variable //

()Using Object.values //
:()Any JavaScript object can be converted to an array using Object.values //

} = const person //
,"name: "John //
,age: 30 //
"city: "New York //
;{ //

;(const myArray = Object.values(person //
:myArray is now a JavaScript array, ready to be displayed //

} = const person //
,"name: "John //
,age: 30 //
"city: "New York //
;{ //

;(const myArray = Object.values(person //
;document.getElementById("demo").innerHTML = myArray //

()Using JSON.stringify //
Any JavaScript object can be stringified (converted to a string) with the JavaScript function //
:()JSON.stringify

} = const person //
,"name: "John //

```
,age: 30   //
"city: "New York   //
;{ //

;(let myString = JSON.stringify(person //
:myString is now a JavaScript string, ready to be displayed //

} = const person //
,"name: "John   //
,age: 30   //
"city: "New York   //
;{ //

;(let myString = JSON.stringify(person //
;document.getElementById("demo").innerHTML = myString //

:The result will be a string following the JSON notation //

{"name":"John","age":50,"city":"New York"} //

.JSON.stringify() is included in JavaScript and supported in all major browsers //

Stringify Dates //
:JSON.stringify converts dates into strings //

} = const person //
,"name: "John   //
()today: new Date   //
;{ //

;(let myString = JSON.stringify(person //
;document.getElementById("demo").innerHTML = myString //

Stringify Functions //
:JSON.stringify will not stringify functions //

} = const person //
,"name: "John   //
{;age: function () {return 30   //
;{ //

;(let myString = JSON.stringify(person //
;document.getElementById("demo").innerHTML = myString //

.This can be "fixed" if you convert the functions into strings before stringifying //

} = const person //
,"name: "John   //
```

{;age: function () {return 30   //
;{ //
;()person.age = person.age.toString //

;(let myString = JSON.stringify(person //
;document.getElementById("demo").innerHTML = myString //

Stringify Arrays //
:It is also possible to stringify JavaScript arrays //

;["const arr = ["John", "Peter", "Sally", "Jane //

;(let myString = JSON.stringify(arr //
;document.getElementById("demo").innerHTML = myString //

:The result will be a string following the JSON notation //

["John","Peter","Sally","Jane"] //

JavaScript Object Accessors //

(JavaScript Accessors (Getters and Setters //
.ECMAScript 5 (ES5 2009) introduced Getter and Setters //

.(Getters and setters allow you to define Object Accessors (Computed Properties //

(JavaScript Getter (The get Keyword //
.This example uses a lang property to get the value of the language property //

:Create an object //
} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
,"language: "en   //
} ()get lang   //
;return this.language   //
{   //
;{ //

:Display data from the object using a getter //
;document.getElementById("demo").innerHTML = person.lang //

(JavaScript Setter (The set Keyword //
.This example uses a lang property to set the value of the language property //

} = const person //
,"firstName: "John   //
,"lastName: "Doe   //

,"" :language   //
} (set lang(lang   //
;this.language = lang   //
{   //
;{ //

:Set an object property using a setter //
;"person.lang = "en //

:Display data from the object //
;document.getElementById("demo").innerHTML = person.language //

?JavaScript Function or Getter //
?What is the differences between these two examples //

} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
} ()fullName: function   //
;return this.firstName + " " + this.lastName   //
{   //
;{ //

:Display data from the object using a method //
;()document.getElementById("demo").innerHTML = person.fullName //

} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
} ()get fullName   //
;return this.firstName + " " + this.lastName   //
{   //
;{ //

:Display data from the object using a getter // //
;document.getElementById("demo").innerHTML = person.fullName //

Data Quality //
.JavaScript can secure better data quality when using getters and setters //

Using the lang property, in this example, returns the value of the language property in //
:upper case

:Create an object //
} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
,"language: "en   //

```
} ()get lang   //
;()return this.language.toUpperCase     //
{   //
;{ //

:Display data from the object using a getter //
;document.getElementById("demo").innerHTML = person.lang //

Using the lang property, in this example, stores an upper case value in the language //
:property

} = const person //
,"firstName: "John   //
,"lastName: "Doe   //
,"" :language   //
} (set lang(lang   //
;()this.language = lang.toUpperCase     //
{   //
;{ //

:Set an object property using a setter //
;"person.lang = "en //

:Display data from the object //
;document.getElementById("demo").innerHTML = person.language //

()Object.defineProperty //
:The Object.defineProperty() method can also be used to add Getters and Setters //

A Counter Example //
Define object //
;{const obj = {counter : 0 //

Define setters and getters //
} ,"Object.defineProperty(obj, "reset //
{;get : function () {this.counter = 0   //
;({ //
} ,"Object.defineProperty(obj, "increment //
{;++get : function () {this.counter   //
;({ //
} ,"Object.defineProperty(obj, "decrement //
{;--get : function () {this.counter   //
;({ //
} ,"Object.defineProperty(obj, "add //
{;set : function (value) {this.counter += value   //
;({ //
} ,"Object.defineProperty(obj, "subtract //
{;set : function (value) {this.counter -= value   //
```

;({ //

:Play with the counter //
;obj.reset //
;obj.add = 5 //
;obj.subtract = 1 //
;obj.increment //
;obj.decrement //

JavaScript Object Constructors //

} (function Person(first, last, age, eye //
;this.firstName = first //
;this.lastName = last //
;this.age = age //
;this.eyeColor = eye //
{ //

Notes //
.It is considered good practice to name constructor functions with an upper-case first letter //

About this //
In a constructor function this does not have a value. It is a substitute for the new object. //
.The value of this will become the new object when a new object is created

(Object Types (Blueprints) (Classes //
.The examples from the previous chapters are limited. They only create single objects //

."Sometimes we need a "blueprint" for creating many objects of the same "type //

.The way to create an "object type", is to use an object constructor function //

.In the example above, function Person() is an object constructor function //

Objects of the same type are created by calling the constructor function with the new //
:keyword

;("const myFather = new Person("John", "Doe", 50, "blue //
;("const myMother = new Person("Sally", "Rally", 48, "green //

:To add a new property to a constructor, you must add it to the constructor function //

} (function Person(first, last, age, eyecolor //
;this.firstName = first //
;this.lastName = last //
;this.age = age //
;this.eyeColor = eyecolor //
;"this.nationality = "English //

{ //

.This way object properties can have default values //

Adding a Method to a Constructor //
:Your constructor function can also define methods //

} (function Person(first, last, age, eyecolor //
;this.firstName = first //
;this.lastName = last //
;this.age = age //
;this.eyeColor = eyecolor //
} ()this.name = function //
;return this.firstName + " " + this.lastName //
;{ //
{ //

You cannot add a new method to an object constructor the same way you add a new //
.method to an existing object

:Adding methods to an object constructor must be done inside the constructor function //

} (function Person(firstName, lastName, age, eyeColor //
;this.firstName = firstName //
;this.lastName = lastName //
;this.age = age //
;this.eyeColor = eyeColor //
} (this.changeName = function (name //
;this.lastName = name //
;{ //
{ //

.The changeName() function assigns the value of name to the person's lastName property //

;("myMother.changeName("Doe //

.JavaScript knows which person you are talking about by "substituting" this with myMother //

Built-in JavaScript Constructors //
:JavaScript has built-in constructors for native objects //

new String() // A new String object //
new Number() // A new Number object //
new Boolean() // A new Boolean object //
new Object() // A new Object object //
new Array() // A new Array object //
new RegExp() // A new RegExp object //
new Function() // A new Function object //

```
new Date()      // A new Date object //
```

The Math() object is not in the list. Math is a global object. The new keyword cannot be // .used on Math

As you can see above, JavaScript has object versions of the primitive data types String, // Number, and Boolean. But there is no reason to create complex objects. Primitive values are :much faster

.()Use string literals "" instead of new String //

.()Use number literals 50 instead of new Number //

.()Use boolean literals true / false instead of new Boolean //

.()Use object literals {} instead of new Object //

.()Use array literals [] instead of new Array //

.()Use pattern literals /()/ instead of new RegExp //

.()Use function expressions () {} instead of new Function //

```
let x1 = "";            // new primitive string //
let x2 = 0;             // new primitive number //
let x3 = false;         // new primitive boolean //

const x4 = {};          // new Object object //
const x5 = [];          // new Array object //
const x6 = /()/         // new RegExp object //
const x7 = function(){}; // new function //
```

String Objects //
"Normally, strings are created as primitives: firstName = "John //

:But strings can also be created as objects using the new keyword //
("firstName = new String("John //

.Learn why strings should not be created as object in the chapter JS Strings //

Number Objects //
Normally, numbers are created as primitives: x = 30 //

:But numbers can also be created as objects using the new keyword //
(x = new Number(30 //

Boolean Objects //
Normally, booleans are created as primitives: x = false //

:But booleans can also be created as objects using the new keyword //
(x = new Boolean(false //

JavaScript Object Prototypes //

.All JavaScript objects inherit properties and methods from a prototype //

:In the previous chapter we learned how to use an object constructor //

} (function Person(first, last, age, eyecolor //
;this.firstName = first //
;this.lastName = last //
;this.age = age //
;this.eyeColor = eyecolor //
{ //

;("const myFather = new Person("John", "Doe", 50, "blue //
;("const myMother = new Person("Sally", "Rally", 48, "green //

Prototype Inheritance //
:All JavaScript objects inherit properties and methods from a prototype //

Date objects inherit from Date.prototype //
Array objects inherit from Array.prototype //
Person objects inherit from Person.prototype //
:The Object.prototype is on the top of the prototype inheritance chain //

.Date objects, Array objects, and Person objects inherit from Object.prototype //

Adding Properties and Methods to Objects //
Sometimes you want to add new properties (or methods) to all existing objects of a given //
.type

.Sometimes you want to add new properties (or methods) to an object constructor //

Using the prototype Property //
:The JavaScript prototype property allows you to add new properties to object constructors //

} (function Person(first, last, age, eyecolor //
;this.firstName = first //
;this.lastName = last //
;this.age = age //
;this.eyeColor = eyecolor //
{ //

;"Person.prototype.nationality = "English //

The JavaScript prototype property also allows you to add new methods to objects //
:constructors

} (function Person(first, last, age, eyecolor //
;this.firstName = first   //
;this.lastName = last   //
;this.age = age   //
;this.eyeColor = eyecolor   //
{ //

} ()Person.prototype.name = function //
;return this.firstName + " " + this.lastName   //
;{ //

JavaScript Iterables //

.Iterable objects are objects that can be iterated over with for..of //

.Technically, iterables must implement the Symbol.iterator method //

Iterating Over a String //
:You can use a for..of loop to iterate over the elements of a string //

} ("for (const x of "W3Schools //
code block to be executed //
{ //

Iterating Over an Array //
:You can use a for..of loop to iterate over the elements of an Array //

} ([for (const x of [1,2,3,4,5 //
code block to be executed //
{ //

JavaScript Iterators //
.The iterator protocol defines how to produce a sequence of values from an object //

.An object becomes an iterator when it implements a next() method //

:The next() method must return an object with two properties //

(value (the next value //
(done (true or false //
The value returned by the iteratorvalue //
(Can be omitted if done is true) //
true if the iterator has completed done //
false if the iterator has produced a new value //

Home Made Iterable //
:This iterable returns never ending: 10,20,30,40,.... Everytime next() is called //

Home Made Iterable //
} ()function myNumbers //
;let n = 0   //
} return   //
} ()next: function   //
;n += 10   //
;{return {value:n, done:false   //
{   //
;{   //
{ //

Create Iterable //
;()const n = myNumbers //
n.next(); // Returns 10 //
n.next(); // Returns 20 //
n.next(); // Returns 30 //

:The problem with a home made iterable //

.It does not support the JavaScript for..of statement //

.A JavaScript iterable is an object that has a Symbol.iterator //

.The Symbol.iterator is a function that returns a next() function //

{ } (An iterable can be iterated over with the code: for (const x of iterable //

Create an Object //
;{} = myNumbers //

Make it Iterable //
} ()myNumbers[Symbol.iterator] = function //
;let n = 0   //
;done = false   //
} return   //
} ()next   //
;n += 10   //
{if (n == 100) {done = true   //
;{return {value:n, done:done   //
{   //
;{   //
{ //
Now you can use for..of //

} (for (const num of myNumbers //

Any Code Here //
{ //

.The Symbol.iterator method is called automatically by for..of //

:"But we can also do it "manually //

;()[let iterator = myNumbers[Symbol.iterator //

} (while (true //
;()const result = iterator.next //
;if (result.done) break //
Any Code Here //
{ //

JavaScript Sets //

.A JavaScript Set is a collection of unique values //

.Each value can only occur once in a Set //

.A Set can hold any value of any data type //

Set Methods //
Description Method //
Creates a new Set new Set() //
Adds a new element to the Set add() //
Removes an element from a Set delete() //
Returns true if a value exists has() //
Removes all elements from a Set clear() //
Invokes a callback for each element forEach() //
Returns an Iterator with all the values in a Set values() //
()Same as values keys() //
Returns an Iterator with the [value,value] pairs from a Set entries() //
Description Property //
Returns the number elements in a Set size //
How to Create a Set //
:You can create a JavaScript Set by //

()Passing an Array to new Set //
Create a new Set and use add() to add values //
Create a new Set and use add() to add variables //
The new Set() Method //
:Pass an Array to the new Set() constructor //

Create a Set //
;(["const letters = new Set(["a","b","c //

:Create a Set and add literal values //

Create a Set //
;()const letters = new Set //

Add Values to the Set //
;("letters.add("a //
;("letters.add("b //
;("letters.add("c //

:Create a Set and add variables //

Create Variables //
;"const a = "a //
;"const b = "b //
;"const c = "c //

Create a Set //
;()const letters = new Set //

Add Variables to the Set //
;(letters.add(a //
;(letters.add(b //
;(letters.add(c //

The add() Method //

;("letters.add("d //
;("letters.add("e //

:If you add equal elements, only the first will be saved //

;("letters.add("a //
;("letters.add("b //
;("letters.add("c //
;("letters.add("c //
;("letters.add("c //
;("letters.add("c //
;("letters.add("c //
;("letters.add("c //

The forEach() Method //
:The forEach() method invokes a function for each Set element //

Create a Set //
;(["const letters = new Set(["a","b","c //

List all entries //

;"" = let text //
} (letters.forEach (function(value //
;text += value //
({ //

The values() Method //
:The values() method returns an Iterator object containing all the values in a Set //

[letters.values() // Returns [object Set Iterator //

:Now you can use the Iterator object to access the elements //

Create an Iterator //

;()const myIterator = letters.values //

List all Values // //
;"" = let text //
} (for (const entry of myIterator //
;text += entry //
{ //

The keys() Method //
.A Set has no keys //

.()keys() returns the same as values //

.This makes Sets compatible with Maps //

[letters.keys() // Returns [object Set Iterator //

The entries() Method //
.A Set has no keys //

.entries() returns [value,value] pairs instead of [key,value] pairs //

:This makes Sets compatible with Maps //

Create an Iterator //
;()const myIterator = letters.entries //

List all Entries //
;"" = let text //
} (for (const entry of myIterator //
;text += entry //
{ //

Sets are Objects //

:For a Set, typeof returns object //

typeof letters; // Returns object //

:For a Set, instanceof Set returns true //

letters instanceof Set; // Returns true //

JavaScript Maps //

.A Map holds key-value pairs where the keys can be any datatype //

.A Map remembers the original insertion order of the keys //

.A Map has a property that represents the size of the map //

Map Methods //
Description Method //
Creates a new Map object new Map() //
Sets the value for a key in a Map set() //
Gets the value for a key in a Map get() //
Removes all the elements from a Map clear() //
Removes a Map element specified by a key delete() //
Returns true if a key exists in a Map has() //
Invokes a callback for each key/value pair in a Map forEach() //
Returns an iterator object with the [key, value] pairs in a Map entries() //
Returns an iterator object with the keys in a Map keys() //
Returns an iterator object of the values in a Map values() //
Description Property //
Returns the number of Map elements size //
How to Create a Map //
:You can create a JavaScript Map by //

()Passing an Array to new Map //
()Create a Map and use Map.set //
()new Map //
:You can create a Map by passing an Array to the new Map() constructor //

Create a Map //
])const fruits = new Map //
,[apples", 500"] //
,[bananas", 300"] //
[oranges", 200"] //
;([//

()Map.set //
:You can add elements to a Map with the set() method //

Create a Map //
;()const fruits = new Map //

Set Map Values //
;(fruits.set("apples", 500 //
;(fruits.set("bananas", 300 //
;(fruits.set("oranges", 200 //

:The set() method can also be used to change existing Map values //

;(fruits.set("apples", 500 //

()Map.get //
:The get() method gets the value of a key in a Map //

fruits.get("apples");   // Returns 500 //

Map.size //
:The size property returns the number of elements in a Map //

;fruits.size //

()Map.delete //
:The delete() method removes a Map element //

;("fruits.delete("apples //

()Map.clear //
:The clear() method removes all the elements from a Map //

;()fruits.clear //

()Map.has //
:The has() method returns true if a key exists in a Map //

;("fruits.has("apples //

:Try This //
;("fruits.delete("apples //
;("fruits.has("apples //

Maps are Objects //
:typeof returns object //

:Returns object //
;typeof fruits //

:instanceof Map returns true //

:Returns true //
fruits instanceof Map //

JavaScript Objects vs Maps //
:Differences between JavaScript Objects and Maps //
Map Object //
Directly iterable Not directly iterable //
Have a size propertyDo not have a size property //
Keys can be any datatype Keys must be Strings (or Symbols) //
Keys are ordered by insertion Keys are not well ordered //
Do not have default keys Have default keys //
()Map.forEach //
:The forEach() method invokes a callback for each key/value pair in a Map //

List all entries //
;"" = let text //
} (fruits.forEach (function(value, key //
;text += key + ' = ' + value //
({ //

()Map.entries //
:The entries() method returns an iterator object with the [key,values] in a Map //

List all entries //
;"" = let text //
} (()for (const x of fruits.entries //
;text += x //
{ //

()Map.keys //
:The keys() method returns an iterator object with the keys in a Map //

List all keys //
;"" = let text //
} (()for (const x of fruits.keys //
;text += x //
{ //

()Map.values //
:The values() method returns an iterator object with the values in a Map //

List all values //
;"" = let text //
} (()for (const x of fruits.values //
;text += x //
{ //

:You can use the values() method to sum the values in a Map //

Sum all values //
;let total = 0 //
} (()for (const x of fruits.values //
;total += x   //
{ //

Objects as Keys //
.Being able to use objects as keys is an important Map feature //

Create Objects //
;{'const apples = {name: 'Apples //
;{'const bananas = {name: 'Bananas //
;{'const oranges = {name: 'Oranges //

Create a Map //
;()const fruits = new Map //

Add new Elements to the Map //
;(fruits.set(apples, 500 //
;(fruits.set(bananas, 300 //
;(fruits.set(oranges, 200 //

:("Remember: The key is an object (apples), not a string ("apples //

JavaScript ES5 Object Methods //

Managing Objects //
Create object with an existing object as prototype //
()Object.create //

Adding or changing an object property //
(Object.defineProperty(object, property, descriptor //

Adding or changing object properties //
(Object.defineProperties(object, descriptors //

Accessing Properties //
(Object.getOwnPropertyDescriptor(object, property //

Returns all properties as an array //
(Object.getOwnPropertyNames(object //

Accessing the prototype //
(Object.getPrototypeOf(object //

Returns enumerable properties as an array //
(Object.keys(object //

Protecting Objects //
Prevents adding properties to an object //
(Object.preventExtensions(object //

Returns true if properties can be added to an object //
(Object.isExtensible(object //

(Prevents changes of object properties (not values //
(Object.seal(object //

Returns true if object is sealed //
(Object.isSealed(object //

Prevents any changes to an object //
(Object.freeze(object //

Returns true if object is frozen //
(Object.isFrozen(object //
Changing a Property Value //
Syntax //
({Object.defineProperty(object, property, {value : value //
:This example changes a property value //

} = const person //
,"firstName: "John //
,"lastName : "Doe //
"language : "EN //
;{ //

Change a property //
{"Object.defineProperty(person, "language", {value : "NO //

Changing Meta Data //
:ES5 allows the following property meta data to be changed //

writable : true // Property value can be changed //
enumerable : true // Property can be enumerated //
configurable : true // Property can be reconfigured //
writable : false // Property value can not be changed //
enumerable : false // Property can be not enumerated //
configurable : false // Property can be not reconfigured //
:ES5 allows getters and setters to be changed //

Defining a getter //
{ get: function() { return language //

Defining a setter //
{ set: function(value) { language = value //
:This example makes language read-only //

;({Object.defineProperty(person, "language", {writable:false //
:This example makes language not enumerable //

;({Object.defineProperty(person, "language", {enumerable:false //
Listing All Properties //
:This example list all properties of an object //

} = const person //
,"firstName: "John //
,"lastName : "Doe //
"language : "EN //
;{ //

;({Object.defineProperty(person, "language", {enumerable:false //
Object.getOwnPropertyNames(person); // Returns an array of properties //

Listing Enumerable Properties //
:This example list only the enumerable properties of an object //

} = const person //
,"firstName: "John //
,"lastName : "Doe //
"language : "EN //
;{ //

;({Object.defineProperty(person, "language", {enumerable:false //
Object.keys(person); // Returns an array of enumerable properties //

Adding a Property //
:This example adds a new property to an object //

:Create an object //
} = const person //
,"firstName: "John //
,"lastName : "Doe //
"language : "EN //
;{ //

Add a property //
;({"Object.defineProperty(person, "year", {value:"2008 //

Adding Getters and Setters //
:The Object.defineProperty() method can also be used to add Getters and Setters //

Create an object//
;{"const person = {firstName:"John", lastName:"Doe //

Define a getter //
} ,"Object.defineProperty(person, "fullName //
{;get: function () {return this.firstName + " " + this.lastName   //
;({ //

A Counter Example //

Define object //
;{const obj = {counter:0 //

Define setters //
} ,"Object.defineProperty(obj, "reset //
{;get : function () {this.counter = 0   //
;({ //
} ,"Object.defineProperty(obj, "increment //
{;++get : function () {this.counter   //
;({ //
} ,"Object.defineProperty(obj, "decrement //
{;--get : function () {this.counter   //
;({ //
} ,"Object.defineProperty(obj, "add //
{;set : function (value) {this.counter += value   //
;({ //
} ,"Object.defineProperty(obj, "subtract //
{;set : function (i) {this.counter -= i   //
;({ //

:Play with the counter //
;obj.reset //
;obj.add = 5 //
;obj.subtract = 1 //
;obj.increment //
;obj.decrement //

JavaScript Function Definitions //

.JavaScript functions are defined with the function keyword //

.You can use a function declaration or a function expression //

Function Declarations //
:Earlier in this tutorial, you learned that functions are declared with the following syntax //

} (function functionName(parameters //

code to be executed //
{ //
Declared functions are not executed immediately. They are "saved for later use", and will //
.(be executed later, when they are invoked (called upon

} (function myFunction(a, b //
;return a * b   //
{ //

Function Expressions //
.A JavaScript function can also be defined using an expression //

:A function expression can be stored in a variable //

;{const x = function (a, b) {return a * b //

After a function expression has been stored in a variable, the variable can be used as a //
:function

;{const x = function (a, b) {return a * b //
;(let z = x(4, 3 //

.(The function above is actually an anonymous function (a function without a name //

Functions stored in variables do not need function names. They are always invoked //
.(called) using the variable name

.The function above ends with a semicolon because it is a part of an executable statement //

The Function() Constructor //
As you have seen in the previous examples, JavaScript functions are defined with the //
.function keyword

Functions can also be defined with a built-in JavaScript function constructor called //
.()Function

;("const myFunction = new Function("a", "b", "return a * b //

;(let x = myFunction(4, 3 //

You actually don't have to use the function constructor. The example above is the same as //
:writing

;{const myFunction = function (a, b) {return a * b //

;(let x = myFunction(4, 3 //

.Most of the time, you can avoid using the new keyword in JavaScript //

Function Hoisting //
.(Earlier in this tutorial, you learned about "hoisting" (JavaScript Hoisting //

Hoisting is JavaScript's default behavior of moving declarations to the top of the current //
.scope

.Hoisting applies to variable declarations and to function declarations //

:Because of this, JavaScript functions can be called before they are declared //

;(myFunction(5 //

} (function myFunction(y //
;return y * y   //
{ //
.Functions defined using an expression are not hoisted //

Self-Invoking Functions //
."Function expressions can be made "self-invoking //

.A self-invoking expression is invoked (started) automatically, without being called //

.() Function expressions will execute automatically if the expression is followed by //

.You cannot self-invoke a function declaration //

You have to add parentheses around the function to indicate that it is a function //
:expression

} () function) //
let x = "Hello!!"; // I will invoke myself   //
;()({ //

The function above is actually an anonymous self-invoking function (function without //
.(name

Functions Can Be Used as Values //
:JavaScript functions can be used as values //

} (function myFunction(a, b //
;return a * b   //
{ //

;(let x = myFunction(4, 3 //

:JavaScript functions can be used in expressions //

```
} (function myFunction(a, b //
;return a * b   //
{ //

;let x = myFunction(4, 3) * 2 //
```

Functions are Objects //
.The typeof operator in JavaScript returns "function" for functions //

.But, JavaScript functions can best be described as objects //

.JavaScript functions have both properties and methods //

The arguments.length property returns the number of arguments received when the //
:function was invoked

```
} (function myFunction(a, b //
;return arguments.length   //
{ //
```

:The toString() method returns the function as a string //

```
} (function myFunction(a, b //
;return a * b   //
{ //

;()let text = myFunction.toString //
```

.A function defined as the property of an object, is called a method to the object //
.A function designed to create new objects, is called an object constructor //

Arrow Functions //
.Arrow functions allows a short syntax for writing function expressions //

.You don't need the function keyword, the return keyword, and the curly brackets //

ES5 //

```
} (var x = function(x, y //
;return x * y   //
{ //
```

ES6 //

```
;const x = (x, y) => x * y //
```

Arrow functions do not have their own this. They are not well suited for defining object //
.methods

.Arrow functions are not hoisted. They must be defined before they are used //

Using const is safer than using var, because a function expression is always constant //
.value

You can only omit the return keyword and the curly brackets if the function is a single //
:statement. Because of this, it might be a good habit to always keep them

;{ const x = (x, y) => { return x * y //

JavaScript Function Parameters //

.(JavaScript function does not perform any checking on parameter values (arguments //

Function Parameters and Arguments //
:Earlier in this tutorial, you learned that functions can have parameters //

} (function functionName(parameter1, parameter2, parameter3 //
code to be executed // //
{ //
.Function parameters are the names listed in the function definition //

.Function arguments are the real values passed to (and received by) the function //

Parameter Rules //
.JavaScript function definitions do not specify data types for parameters //

.JavaScript functions do not perform type checking on the passed arguments //

.JavaScript functions do not check the number of arguments received //

Default Parameters //
If a function is called with missing arguments (less than declared), the missing values are //
.set to undefined

Sometimes this is acceptable, but sometimes it is better to assign a default value to the //
:parameter

} (function myFunction(x, y //
} (if (y === undefined //
;y = 2 //
{ //
{ //

Default Parameter Values //

.If y is not passed or undefined, then y = 10 //

} (function myFunction(x, y = 10 //

```
;return x + y   //
{ //
;(myFunction(5 //
```

Function Rest Parameter //
The rest parameter (...) allows a function to treat an indefinite number of arguments as an //
:array

```
} (function sum(...args //
;let sum = 0   //
;for (let arg of args) sum += arg   //
;return sum   //
{ //

;(let x = sum(4, 9, 16, 25, 29, 100, 66, 77 //
```

The Arguments Object //
.JavaScript functions have a built-in object called the arguments object //

The argument object contains an array of the arguments used when the function was //
.(called (invoked

This way you can simply use a function to find (for instance) the highest value in a list of //
:numbers

```
;(x = findMax(1, 123, 500, 115, 44, 88 //

} ()function findMax //
;let max = -Infinity   //
} (++for (let i = 0; i < arguments.length; i   //
} (if (arguments[i] > max     //
;[max = arguments[i       //
{     //
{   //
;return max   //
{ //
```

:Or create a function to sum all input values //

```
;(x = sumAll(1, 123, 500, 115, 44, 88 //

} ()function sumAll //
;let sum = 0   //
} (++for (let i = 0; i < arguments.length; i   //
;[sum += arguments[i     //
{   //
;return sum   //
{ //
```

JavaScript Function Invocation //

.The code inside a JavaScript function will execute when "something" invokes it //

Invoking a JavaScript Function //
.The code inside a function is not executed when the function is defined //

.The code inside a function is executed when the function is invoked //

."It is common to use the term "call a function" instead of "invoke a function //

."It is also common to say "call upon a function", "start a function", or "execute a function //

In this tutorial, we will use invoke, because a JavaScript function can be invoked without //
.being called

Invoking a Function as a Function //

} (function myFunction(a, b //
;return a * b   //
{ //
myFunction(10, 2);   // Will return 20 //

The function above does not belong to any object. But in JavaScript there is always a //
.default global object

In HTML the default global object is the HTML page itself, so the function above "belongs" //
.to the HTML page

In a browser the page object is the browser window. The function above automatically //
.becomes a window function

Note //
.This is a common way to invoke a JavaScript function, but not a very good practice //
Global variables, methods, or functions can easily create name conflicts and bugs in the //
.global object

:myFunction() and window.myFunction() is the same function //

} (function myFunction(a, b //
;return a * b   //
{ //
window.myFunction(10, 2);   // Will also return 20 //

The Global Object //
When a function is called without an owner object, the value of this becomes the global //
.object

.In a web browser the global object is the browser window //

:This example returns the window object as the value of this //

let x = myFunction();   // x will be the window object //

} ()function myFunction //
;return this   //
{ //

nvoking a Function as a Method //
.In JavaScript you can define functions as object methods //

The following example creates an object (myObject), with two properties (firstName and //
:(lastName), and a method (fullName

} = const myObject //
,"firstName:"John   //
,"lastName: "Doe   //
} () fullName: function   //
;return this.firstName + " " + this.lastName   //
{   //
{ //
"myObject.fullName();   // Will return "John Doe //

The fullName method is a function. The function belongs to the object. myObject is the //
.owner of the function

The thing called this, is the object that "owns" the JavaScript code. In this case the value of //
.this is myObject

:Test it! Change the fullName method to return the value of this //

} = const myObject //
,"firstName:"John   //
,"lastName: "Doe   //
} () fullName: function   //
;return this   //
{   //
{ //

(This will return [object Object] (the owner object //
;()myObject.fullName //

Invoking a Function with a Function Constructor //
.If a function invocation is preceded with the new keyword, it is a constructor invocation //

It looks like you create a new function, but since JavaScript functions are objects you //
:actually create a new object

:This is a function constructor //
} (function myFunction(arg1, arg2 //
;this.firstName = arg1   //
;this.lastName = arg2   //
{ //

This creates a new object //
;("const myObj = new myFunction("John", "Doe //

"This will return "John //
;myObj.firstName //

()JavaScript Function call //

Method Reuse //
.With the call() method, you can write a method that can be used on different objects //

All Functions are Methods //
.In JavaScript all functions are object methods //

If a function is not a method of a JavaScript object, it is a function of the global object (see //
.(previous chapter

.The example below creates an object with 3 properties, firstName, lastName, fullName //

} = const person //
,"firstName:"John   //
,"lastName: "Doe   //
} () fullName: function   //
;return this.firstName + " " + this.lastName   //
{   //
{ //

:"This will return "John Doe //
;()person.fullName //

.In the example above, this refers to the person object //

.this.firstName means the firstName property of this //

:Same as //

.this.firstName means the firstName property of person //

The JavaScript call() Method //

.The call() method is a predefined JavaScript method //

.(It can be used to invoke (call) a method with an owner object as an argument (parameter //

.With call(), an object can use a method belonging to another object //

:This example calls the fullName method of person, using it on person1 //

```
} = const person //
} ()fullName: function   //
;return this.firstName + " " + this.lastName     //
{   //
{ //
} = const person1 //
,"firstName:"John   //
"lastName: "Doe   //
{ //
} = const person2 //
,"firstName:"Mary   //
"lastName: "Doe   //
{ //
```

:"This will return "John Doe //
;(person.fullName.call(person1 //

:This example calls the fullName method of person, using it on person2 //

```
} = const person //
} ()fullName: function   //
;return this.firstName + " " + this.lastName     //
{   //
{ //
} = const person1 //
,"firstName:"John   //
"lastName: "Doe   //
{ //
} = const person2 //
,"firstName:"Mary   //
"lastName: "Doe   //
{ //
```

"This will return "Mary Doe //
;(person.fullName.call(person2 //

The call() Method with Arguments //
:The call() method can accept arguments //

```
} = const person //
```

} (fullName: function(city, country   //
;return this.firstName + " " + this.lastName + "," + city + "," + country   //
{   //
{ //

} = const person1 //
,"firstName:"John   //
"lastName: "Doe   //
{ //

;("person.fullName.call(person1, "Oslo", "Norway //

()JavaScript Function apply //

Method Reuse //
.With the apply() method, you can write a method that can be used on different objects //

The JavaScript apply() Method //
.(The apply() method is similar to the call() method (previous chapter //

:In this example the fullName method of person is applied on person1 //

} = const person //
} ()fullName: function   //
;return this.firstName + " " + this.lastName   //
{   //
{ //

} = const person1 //
,"firstName: "Mary   //
"lastName: "Doe   //
{ //

:"This will return "Mary Doe //
;(person.fullName.apply(person1 //

()he Difference Between call() and apply //
:The difference is //

.The call() method takes arguments separately //

.The apply() method takes arguments as an array //

.The apply() method is very handy if you want to use an array instead of an argument list //

The apply() Method with Arguments //
:The apply() method accepts arguments in an array //

```
} = const person //
} (fullName: function(city, country   //
;return this.firstName + " " + this.lastName + "," + city + "," + country     //
{   //
{ //

} = const person1 //
,"firstName:"John   //
"lastName: "Doe   //
{ //

;(["person.fullName.apply(person1, ["Oslo", "Norway //

:Compared with the call() method //

} = const person //
} (fullName: function(city, country   //
;return this.firstName + " " + this.lastName + "," + city + "," + country     //
{   //
{ //

} = const person1 //
,"firstName:"John   //
"lastName: "Doe   //
{ //

;("person.fullName.call(person1, "Oslo", "Norway //

Simulate a Max Method on Arrays //
:You can find the largest number (in a list of numbers) using the Math.max() method //

Math.max(1,2,3);  // Will return 3 //

Since JavaScript arrays do not have a max() method, you can apply the Math.max() //
.method instead

Math.max.apply(null, [1,2,3]); // Will also return //

.The first argument (null) does not matter. It is not used in this example //

:These examples will give the same result //

Math.max.apply(Math, [1,2,3]); // Will also return 3 //

Math.max.apply(" ", [1,2,3]); // Will also return 3 //

Math.max.apply(0, [1,2,3]); // Will also return 3 //
```

JavaScript Strict Mode //
In JavaScript strict mode, if the first argument of the apply() method is not an object, it //
becomes the owner (object) of the invoked function. In "non-strict" mode, it becomes the
.global object

()JavaScript Function bind //

Function Borrowing //
.With the bind() method, an object can borrow a method from another object //

.(The example below creates 2 objects (person and member //

:The member object borrows the fullname method from the person object //

} = const person //
,"firstName:"John //
,"lastName: "Doe //
} () fullName: function //
;return this.firstName + " " + this.lastName //
{ //
{ //

} = const member //
,"firstName:"Hege //
,"lastName: "Nilsen //
{ //

;(let fullName = person.fullName.bind(member //

Preserving this //
.Sometimes the bind() method has to be used to prevent losing this //

In the following example, the person object has a display method. In the display method, //
:this refers to the person object

} = const person //
,"firstName:"John //
,"lastName: "Doe //
} () display: function //
;("let x = document.getElementById("demo //
;x.innerHTML = this.firstName + " " + this.lastName //
{ //
{ //

;()person.display //

.When a function is used as a callback, this is lost //

This example will try to display the person name after 3 seconds, but it will display //
:undefined instead

} = const person //
,"firstName:"John //
,"lastName: "Doe //
} () display: function //
;("let x = document.getElementById("demo //
;x.innerHTML = this.firstName + " " + this.lastName //
{ //
{ //

;(setTimeout(person.display, 3000 //

.The bind() method solves this problem //

.In the following example, the bind() method is used to bind person.display to person //

:This example will display the person name after 3 seconds //

} = const person //
,"firstName:"John //
,"lastName: "Doe //
} () display: function //
;("let x = document.getElementById("demo //
;x.innerHTML = this.firstName + " " + this.lastName //
{ //
{ //

;(let display = person.display.bind(person //
;(setTimeout(display, 3000 //

JavaScript Closures //

.JavaScript variables can belong to the local or global scope //

.Global variables can be made local (private) with closures //

Global Variables //
:A function can access all variables defined inside the function, like this //

} ()function myFunction //
;let a = 4 //
;return a * a //
{ //

:But a function can also access variables defined outside the function, like this //

```
;let a = 4 //
} ()function myFunction //
;return a * a   //
{ //
```

.In the last example, a is a global variable //

.In a web page, global variables belong to the page //

.Global variables can be used (and changed) by all other scripts in the page //

.In the first example, a is a local variable //

A local variable can only be used inside the function where it is defined. It is hidden from // .other functions and other scripting code

Global and local variables with the same name are different variables. Modifying one, does // .not modify the other

Note //
Variables created without a declaration keyword (var, let, or const) are always global, even // .if they are created inside a function

```
} ()function myFunction //
;a = 4   //
{ //
```

Variable Lifetime //
Global variables live until the page is discarded, like when you navigate to another page or // .close the window

Local variables have short lives. They are created when the function is invoked, and // .deleted when the function is finished

A Counter Dilemma //
Suppose you want to use a variable for counting something, and you want this counter to // .be available to all functions

:You could use a global variable, and a function to increase the counter //

Initiate counter //

```
;let counter = 0 //
```

Function to increment counter //

```
} ()function add //
;counter += 1   //
{ //
```

Call add() 3 times //
;()add //
;()add //
;()add //

The counter should now be 3 //

There is a problem with the solution above: Any code on the page can change the counter, //
.()without calling add

:The counter should be local to the add() function, to prevent other code from changing it //

Initiate counter //
;let counter = 0 //

Function to increment counter //
} ()function add //
;let counter = 0   //
;counter += 1   //
{ //

Call add() 3 times //
;()add //
;()add //
;()add //

The counter should now be 3. But it is 0//

.It did not work because we display the global counter instead of the local counter //

We can remove the global counter and access the local counter by letting the function //
:return it

.t did not work because we reset the local counter every time we call the function //

.A JavaScript inner function can solve this //

JavaScript Nested Functions //
.All functions have access to the global scope //

.In fact, in JavaScript, all functions have access to the scope "above" them //

JavaScript supports nested functions. Nested functions have access to the scope "above" //
.them

In this example, the inner function plus() has access to the counter variable in the parent //
:function

} ()function add //
;let counter = 0 //
{;function plus() {counter += 1 //
;()plus //
;return counter //
{ //

This could have solved the counter dilemma, if we could reach the plus() function from the //
.outside

.We also need to find a way to execute counter = 0 only once //

.We need a closure //

JavaScript Closures //
?Remember self-invoking functions? What does this function do //

} () const add = (function //
;let counter = 0 //
{return function () {counter += 1; return counter //
;()({ //

;()add //
;()add //
;()add //

the counter is now 3 //

Example Explained //
.The variable add is assigned to the return value of a self-invoking function //

The self-invoking function only runs once. It sets the counter to zero (0), and returns a //
.function expression

This way add becomes a function. The "wonderful" part is that it can access the counter in //
.the parent scope

This is called a JavaScript closure. It makes it possible for a function to have "private" //
.variables

The counter is protected by the scope of the anonymous function, and can only be //
.changed using the add function

JavaScript Classes //

avaScript Class Syntax //
.Use the keyword class to create a class //

:()Always add a method named constructor //

Syntax //
} class ClassName //
{ ... } ()constructor //
{ //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
{ //

."The example above creates a class named "Car //

."The class has two initial properties: "name" and "year //

.A JavaScript class is not an object //

.It is a template for JavaScript objects //

Using a Class //
:When you have a class, you can use the class to create objects //

;(const myCar1 = new Car("Ford", 2014 //
;(const myCar2 = new Car("Audi", 2019 //

.The example above uses the Car class to create two Car objects //

.The constructor method is called automatically when a new object is created //

The Constructor Method //
:The constructor method is a special method //

"It has to have the exact name "constructor //
It is executed automatically when a new object is created //
It is used to initialize object properties //
If you do not define a constructor method, JavaScript will add an empty constructor //
.method

Class Methods //
.Class methods are created with the same syntax as object methods //

.Use the keyword class to create a class //

.Always add a constructor() method //

.Then add any number of methods //

Syntax //
} class ClassName //
{ ... } ()constructor //
{ ... } ()method_1 //
{ ... } ()method_2 //
{ ... } ()method_3 //
{ //
:Create a Class method named "age", that returns the Car age //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
} ()age //
;()const date = new Date //
;return date.getFullYear() - this.year //
{ //
{ //

;(const myCar = new Car("Ford", 2014 //
= document.getElementById("demo").innerHTML //
;".My car is " + myCar.age() + " years old" //

"use strict" //
."The syntax in classes must be written in "strict mode //

.You will get an error if you do not follow the "strict mode" rules //

:In "strict mode" you will get an error if you use a variable without declaring it //

} class Car //
} (constructor(name, year //
;this.name = name //
;this.year = year //
{ //
} ()age //
date = new Date(); // This will not work // //
const date = new Date(); // This will work //
;return date.getFullYear() - this.year //
{ //
{ //

JavaScript Class Inheritance //

Class Inheritance //

.To create a class inheritance, use the extends keyword //

:A class created with a class inheritance inherits all the methods from another class //

:Create a class named "Model" which will inherit the methods from the "Car" class //

```
} class Car //
} (constructor(brand   //
;this.carname = brand     //
{   //
} ()present   //
;return 'I have a ' + this.carname     //
{   //
{ //

} class Model extends Car //
} (constructor(brand, mod   //
;(super(brand     //
;this.model = mod     //
{   //
} ()show   //
;return this.present() + ', it is a ' + this.model     //
{   //
{ //

;("let myCar = new Model("Ford", "Mustang //
;()document.getElementById("demo").innerHTML = myCar.show //
```

.The super() method refers to the parent class //

By calling the super() method in the constructor method, we call the parent's constructor //
.method and gets access to the parent's properties and methods

Getters and Setters //
.Classes also allows you to use getters and setters //

It can be smart to use getters and setters for your properties, especially if you want to do //
.something special with the value before returning them, or before you set them

To add getters and setters in the class, use the get and set keywords //

:Create a getter and a setter for the "carname" property //

```
} class Car //
} (constructor(brand   //
;this.carname = brand     //
{   //
} ()get cnam   //
```

```
;return this.carname     //
{   //
} (set cnam(x   //
;this.carname = x     //
{   //
{ //

;("const myCar = new Car("Ford //

;document.getElementById("demo").innerHTML = myCar.cnam //
```

Note: even if the getter is a method, you do not use parentheses when you want to get the //
.property value

The name of the getter/setter method cannot be the same as the name of the property, in //
.this case carname

Many programmers use an underscore character _ before the property name to separate //
:the getter/setter from the actual property

You can use the underscore character to separate the getter/setter from the actual //
:property

```
} class Car //
} (constructor(brand   //
;this._carname = brand     //
{   //
} ()get carname   //
;return this._carname     //
{   //
} (set carname(x   //
;this._carname = x     //
{   //
{ //

;("const myCar = new Car("Ford //

;document.getElementById("demo").innerHTML = myCar.carname //
```

To use a setter, use the same syntax as when you set a property value, without //
:parentheses

```
:"Use a setter to change the carname to "Volvo //

} class Car //
} (constructor(brand   //
;this._carname = brand     //
{   //
```

```
} ()get carname   //
;return this._carname     //
{   //
} (set carname(x   //
;this._carname = x     //
{   //
{ //

;("const myCar = new Car("Ford //
;"myCar.carname = "Volvo //
;document.getElementById("demo").innerHTML = myCar.carname //
```

Hoisting //
.Unlike functions, and other JavaScript declarations, class declarations are not hoisted //

:That means that you must declare a class before you can use it //

.You cannot use the class yet//
.myCar = new Car("Ford") will raise an error//

```
} class Car //
} (constructor(brand   //
;this.carname = brand     //
{   //
{ //
```

:Now you can use the class//
```
("const myCar = new Car("Ford //
```

JavaScript Static Methods //

.Static class methods are defined on the class itself //

.You cannot call a static method on an object, only on an object class //

```
} class Car //
} (constructor(name   //
;this.name = name     //
{   //
} ()static hello   //
;"!!return "Hello     //
{   //
{ //

;("const myCar = new Car("Ford //
```

:You can call 'hello()' on the Car Class //
```
;()document.getElementById("demo").innerHTML = Car.hello //
```

:But NOT on a Car Object //
;()document.getElementById("demo").innerHTML = myCar.hello //
.this will raise an error //

If you want to use the myCar object inside the static method, you can send it as a //
:parameter

} class Car //
} (constructor(name //
;this.name = name //
{ //
} (static hello(x //
;return "Hello " + x.name //
{ //
{ //
;("const myCar = new Car("Ford //
;(document.getElementById("demo").innerHTML = Car.hello(myCar //

JavaScript Callbacks //

"!I will call back later" //

A callback is a function passed as an argument to another function //

This technique allows a function to call another function //

A callback function can run after another function has finished //

Function Sequence //
JavaScript functions are executed in the sequence they are called. Not in the sequence //
.they are defined

:"This example will end up displaying "Goodbye //

} ()function myFirst //
;("myDisplayer("Hello //
{ //

} ()function mySecond //
;("myDisplayer("Goodbye //
{ //

;()myFirst //
;()mySecond //

:"This example will end up displaying "Hello //

} ()function myFirst //
;("myDisplayer("Hello   //
{ //

} ()function mySecond //
;("myDisplayer("Goodbye   //
{ //

;()mySecond //

Sequence Control //
.Sometimes you would like to have better control over when to execute a function //

.Suppose you want to do a calculation, and then display the result //

You could call a calculator function (myCalculator), save the result, and then call another //
:function (myDisplayer) to display the result

} (function myDisplayer(some //
;document.getElementById("demo").innerHTML = some   //
{ //

} (function myCalculator(num1, num2 //
;let sum = num1 + num2   //
;return sum   //
{ //

;(let result = myCalculator(5, 5 //
;(myDisplayer(result //

Or, you could call a calculator function (myCalculator), and let the calculator function call //
:(the display function (myDisplayer

} (function myDisplayer(some //
;document.getElementById("demo").innerHTML = some   //
{ //

} (function myCalculator(num1, num2 //
;let sum = num1 + num2   //
;(myDisplayer(sum   //
{ //

;(myCalculator(5, 5 //

The problem with the first example above, is that you have to call two functions to display //
.the result

The problem with the second example, is that you cannot prevent the calculator function //
.from displaying the result

.Now it is time to bring in a callback //

JavaScript Callbacks //
.A callback is a function passed as an argument to another function //

Using a callback, you could call the calculator function (myCalculator) with a callback //
:(myCallback), and let the calculator function run the callback after the calculation is finished

} (function myDisplayer(some //
;document.getElementById("demo").innerHTML = some //
{ //

} (function myCalculator(num1, num2, myCallback //
;let sum = num1 + num2 //
;(myCallback(sum //
{ //

;(myCalculator(5, 5, myDisplayer //

.In the example above, myDisplayer is a called a callback function //

.It is passed to myCalculator() as an argument //

Note //
.When you pass a function as an argument, remember not to use parenthesis //

;(Right: myCalculator(5, 5, myDisplayer //

;(()Wrong: myCalculator(5, 5, myDisplayer //

Create an Array //
;[const myNumbers = [4, 1, -20, -7, 5, 9, -6 //

Call removeNeg with a callback //
;(const posNumbers = removeNeg(myNumbers, (x) => x >= 0 //

Display Result //
;document.getElementById("demo").innerHTML = posNumbers //

Keep only positive numbers //
} (function removeNeg(numbers, callback //
;[] = const myArray //
} (for (const x of numbers //
} ((if (callback(x //
;(myArray.push(x //

{   //
{   //
;return myArray   //
{ //

.In the example above, (x) => x >= 0 is a callback function //

.It is passed to removeNeg() as an argument //

?When to Use a Callback //
.The examples above are not very exciting //

.They are simplified to teach you the callback syntax //

Where callbacks really shine are in asynchronous functions, where one function has to //
.(wait for another function (like waiting for a file to load

.Asynchronous functions are covered in the next chapter //

Asynchronous JavaScript //

"!I will finish later" //

Functions running in parallel with other functions are called asynchronous //

()A good example is JavaScript setTimeout //

Asynchronous JavaScript //
.The examples used in the previous chapter, was very simplified //

:The purpose of the examples was to demonstrate the syntax of callback functions //

} (function myDisplayer(something //
;document.getElementById("demo").innerHTML = something   //
{ //

} (function myCalculator(num1, num2, myCallback //
;let sum = num1 + num2   //
;(myCallback(sum   //
{ //

;(myCalculator(5, 5, myDisplayer //

.In the example above, myDisplayer is the name of a function //

.It is passed to myCalculator() as an argument //

.In the real world, callbacks are most often used with asynchronous functions //

.()A typical example is JavaScript setTimeout //

Waiting for a Timeout //
When using the JavaScript function setTimeout(), you can specify a callback function to be //
:executed on time-out

;(setTimeout(myFunction, 3000 //

} ()function myFunction //
;"!! document.getElementById("demo").innerHTML = "I love You //
{ //

.In the example above, myFunction is used as a callback //

.myFunction is passed to setTimeout() as an argument //

is the number of milliseconds before time-out, so myFunction() will be called after 3 3000 //
.seconds

Note //
.When you pass a function as an argument, remember not to use parenthesis //

;(Right: setTimeout(myFunction, 3000 //

;(Wrong: setTimeout(myFunction(), 3000 //

Instead of passing the name of a function as an argument to another function, you can //
:always pass a whole function instead

;(setTimeout(function() { myFunction("I love You !!!"); }, 3000 //

} (function myFunction(value //
;document.getElementById("demo").innerHTML = value //
{ //

In the example above, function(){ myFunction("I love You !!!"); } is used as a callback. It is //
.a complete function. The complete function is passed to setTimeout() as an argument

is the number of milliseconds before time-out, so myFunction() will be called after 3 3000 //
.seconds

:Waiting for Intervals //
When using the JavaScript function setInterval(), you can specify a callback function to be //
:executed for each interval

;(setInterval(myFunction, 1000 //

} ()function myFunction //
;()let d = new Date //
=document.getElementById("demo").innerHTML //
+ ":" + ()d.getHours //
+ ":" + ()d.getMinutes //
;()d.getSeconds //
{ //

.In the example above, myFunction is used as a callback //

.myFunction is passed to setInterval() as an argument //

is the number of milliseconds between intervals, so myFunction() will be called every 1000 //
.second

Callback Alternatives //
With asynchronous programming, JavaScript programs can start long-running tasks, and //
.continue running other tasks in paralell

.But, asynchronus programmes are difficult to write and difficult to debug //

Because of this, most modern asynchronous JavaScript methods don't use callbacks. //
.Instead, in JavaScript, asynchronous programming is solved using Promises instead

Note //
.You will learn about promises in the next chapter of this tutorial //

JavaScript Promises //

"!I Promise a Result" //

Producing code" is code that can take some time" //

Consuming code" is code that must wait for the result" //

A Promise is a JavaScript object that links producing code and consuming code //

JavaScript Promise Object //
A JavaScript Promise object contains both the producing code and calls to the consuming //
:code

Promise Syntax //
} (let myPromise = new Promise(function(myResolve, myReject //
(Producing Code" (May take some time" // //

myResolve(); // when successful //
myReject(); // when error //
;({ //

(Consuming Code" (Must wait for a fulfilled Promise" // //
)myPromise.then //
,{ /* function(value) { /* code if successful //
{ /* function(error) { /* code if some error //
;(//
:When the producing code obtains the result, it should call one of the two callbacks //

Call Result //
(myResolve(result value Success //
(myReject(error object Error //
Promise Object Properties //
:A JavaScript Promise object can be //

Pending //
Fulfilled //
Rejected //
.The Promise object supports two properties: state and result //

.While a Promise object is "pending" (working), the result is undefined //

.When a Promise object is "fulfilled", the result is a value //

.When a Promise object is "rejected", the result is an error object //

myPromise.result myPromise.state //
undefined pending"" //
a result value fulfilled"" //
an error object rejected"" //
.You cannot access the Promise properties state and result //

.You must use a Promise method to handle promises //

Promise How To //
:Here is how to use a Promise //

)myPromise.then //
,{ /* function(value) { /* code if successful //
{ /* function(error) { /* code if some error //
;(//
.Promise.then() takes two arguments, a callback for success and another for failure //

Both are optional, so you can add a callback for success or failure only //

} (function myDisplayer(some //
;document.getElementById("demo").innerHTML = some //
{ //

```
} (let myPromise = new Promise(function(myResolve, myReject //
;let x = 0   //

(The producing code (this may take some time // //

} (if (x == 0   //
;("myResolve("OK     //
} else {   //
;("myReject("Error     //
{   //
;({ //

)myPromise.then //
,{;(function(value) {myDisplayer(value   //
{;(function(error) {myDisplayer(error   //
;( //
```

JavaScript Promise Examples //
To demonstrate the use of promises, we will use the callback examples from the previous //
:chapter

Waiting for a Timeout //
Waiting for a File //
Waiting for a Timeout //
Example Using Callback //

```
;(setTimeout(function() { myFunction("I love You !!!"); }, 3000 //

} (function myFunction(value //
;document.getElementById("demo").innerHTML = value   //
{ //
```

Example Using Promise //

```
} (let myPromise = new Promise(function(myResolve, myReject //
;(setTimeout(function() { myResolve("I love You !!"); }, 3000   //
;({ //

} (myPromise.then(function(value //
;document.getElementById("demo").innerHTML = value   //
;({ //
```

Waiting for a file //

```
} (function getFile(myCallback //
;()let req = new XMLHttpRequest   //
;("req.open('GET', "mycar.html   //
} ()req.onload = function   //
} (if (req.status == 200     //
;(myCallback(req.responseText       //
```

```
} else {     //
;(myCallback("Error: " + req.status       //
{     //
{   //
;()req.send   //
{ //

;(getFile(myDisplayer //

} (let myPromise = new Promise(function(myResolve, myReject //
;()let req = new XMLHttpRequest   //
;("req.open('GET', "mycar.htm   //
} ()req.onload = function   //
} (if (req.status == 200     //
;(myResolve(req.response       //
} else {     //
;("myReject("File not Found       //
{     //
;{   //
;()req.send   //
;({ //

)myPromise.then //
,{;(function(value) {myDisplayer(value   //
{;(function(error) {myDisplayer(error   //
;( //

JavaScript Async //

"async and await make promises easier to write" //

async makes a function return a Promise //

await makes a function wait for a Promise //

Async Syntax //
:The keyword async before a function makes the function return a promise //

} ()async function myFunction //
;"return "Hello   //
{ //
:Is the same as //

} ()function myFunction //
;("return Promise.resolve("Hello   //
{ //

:Here is how to use the Promise //
```

```
)myFunction().then //
,{ /* function(value) { /* code if successful   //
{ /* function(error) { /* code if some error   //
;( //

} ()async function myFunction //
;"return "Hello   //
{ //
)myFunction().then //
,{;(function(value) {myDisplayer(value   //
{;(function(error) {myDisplayer(error   //
;( //

:(Or simpler, since you expect a normal value (a normal response, not an error //

} ()async function myFunction //
;"return "Hello   //
{ //
)myFunction().then //
{;(function(value) {myDisplayer(value   //
;( //

Await Syntax //
.The await keyword can only be used inside an async function //

The await keyword makes the function pause the execution and wait for a resolved //
:promise before it continues

;let value = await promise //

Example //
.Let's go slowly and learn how to use it //

Basic Syntax //
} ()async function myDisplay //
} (let myPromise = new Promise(function(resolve, reject   //
;("!! resolve("I love You     //
;({   //
;document.getElementById("demo").innerHTML = await myPromise   //
{ //

;()myDisplay //

Example without reject //
} ()async function myDisplay //
} (let myPromise = new Promise(function(resolve   //
;("!! resolve("I love You     //
```

```
;({   //
;document.getElementById("demo").innerHTML = await myPromise   //
{ //

;()myDisplay //

Waiting for a Timeout //
} ()async function myDisplay //
} (let myPromise = new Promise(function(resolve   //
;(setTimeout(function() {resolve("I love You !!");}, 3000     //
;({   //
;document.getElementById("demo").innerHTML = await myPromise   //
{ //

;()myDisplay //

Waiting for a File //
} ()async function getFile //
} (let myPromise = new Promise(function(resolve   //
;()let req = new XMLHttpRequest     //
;("req.open('GET', "mycar.html     //
} ()req.onload = function     //
} (if (req.status == 200       //
;(resolve(req.response         //
} else {       //
;("resolve("File not Found         //
{       //
;{     //
;()req.send     //
;({   //
;document.getElementById("demo").innerHTML = await myPromise   //
{ //

;()getFile //

JavaScript HTML DOM //

With the HTML DOM, JavaScript can access and change all the elements of an HTML //
.document

(The HTML DOM (Document Object Model //
.When a web page is loaded, the browser creates a Document Object Model of the page //

:The HTML DOM model is constructed as a tree of Objects //

:With the object model, JavaScript gets all the power it needs to create dynamic HTML //

JavaScript can change all the HTML elements in the page //
```

JavaScript can change all the HTML attributes in the page //
JavaScript can change all the CSS styles in the page //
JavaScript can remove existing HTML elements and attributes //
JavaScript can add new HTML elements and attributes //
JavaScript can react to all existing HTML events in the page //
JavaScript can create new HTML events in the page //
What You Will Learn //
:In the next chapters of this tutorial you will learn //

How to change the content of HTML elements //
How to change the style (CSS) of HTML elements //
How to react to HTML DOM events //
How to add and delete HTML elements //

JavaScript - HTML DOM Methods //

.(HTML DOM methods are actions you can perform (on HTML Elements //

.HTML DOM properties are values (of HTML Elements) that you can set or change //

The DOM Programming Interface //
The HTML DOM can be accessed with JavaScript (and with other programming //
.(languages

.In the DOM, all HTML elements are defined as objects //

.The programming interface is the properties and methods of each object //

A property is a value that you can get or set (like changing the content of an HTML //
.(element

.(A method is an action you can do (like add or deleting an HTML element //

The following example changes the content (the innerHTML) of the element with //
:"id="demo

<html> */}
<body>

<script>
;"!document.getElementById("demo").innerHTML = "Hello World
<script/>

<body/>
{/* <html/>

.In the example above, getElementById is a method, while innerHTML is a property //

The getElementById Method //
.The most common way to access an HTML element is to use the id of the element //

.In the example above the getElementById method used id="demo" to find the element //

The innerHTML Property //
.The easiest way to get the content of an element is by using the innerHTML property //

.The innerHTML property is useful for getting or replacing the content of HTML elements //

The innerHTML property can be used to get or change any HTML element, including //
.<<html> and <body

JavaScript HTML DOM Document //

.The HTML DOM document object is the owner of all other objects in your web page //

The HTML DOM Document Object //
.The document object represents your web page //

If you want to access any element in an HTML page, you always start with accessing the //
.document object

Below are some examples of how you can use the document object to access and //
.manipulate HTML

Finding HTML Elements //
Description Method //
Find an element by element id document.getElementById(id) //
Find elements by tag name document.getElementsByTagName(name) //
Find elements by class name document.getElementsByClassName(name) //
Changing HTML Elements //
Description Property //
Change the inner HTML of an element element.innerHTML = new html content //
Change the attribute value of an HTML element element.attribute = new value //
Change the style of an HTML element element.style.property = new style //
Description Method //
Change the attribute value of an HTML element element.setAttribute(attribute, value) //
Adding and Deleting Elements //
Description Method //
Create an HTML element document.createElement(element) //
Remove an HTML element document.removeChild(element) //
Add an HTML element document.appendChild(element) //
Replace an HTML element document.replaceChild(new, old) //
Write into the HTML output stream document.write(text) //

Adding Events Handlers //
Description Method //
Adding event handler code to an document.getElementById(id).onclick = function(){code} //
onclick event
Finding HTML Objects //
The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and //
.properties. These are still valid in HTML5

.Later, in HTML DOM Level 3, more objects, collections, and properties were added //

DOM Description Property //
1 Returns all <a> elements that have a name attribute document.anchors //
1 Deprecated document.applets //
3 Returns the absolute base URI of the document document.baseURI //
1 Returns the <body> element document.body //
1 Returns the document's cookie document.cookie //
3 Returns the document's doctype document.doctype //
3 Returns the <html> element document.documentElement //
3 Returns the mode used by the browser document.documentMode //
3 Returns the URI of the document document.documentURI //
1 Returns the domain name of the document server document.domain //
3 Obsolete. document.domConfig //
3 Returns all <embed> elements document.embeds //
1 Returns all <form> elements document.forms //
3 Returns the <head> element document.head //
1 Returns all <img> elements document.images //
3 Returns the DOM implementation document.implementation //
3 Returns the document's encoding (character set) document.inputEncoding //
3 Returns the date and time the document was updated document.lastModified //
1 Returns all <area> and <a> elements that have a href attribute document.links //
3 Returns the (loading) status of the document document.readyState //
1 Returns the URI of the referrer (the linking document) document.referrer //
3 Returns all <script> elements document.scripts //
3 Returns if error checking is enforced document.strictErrorChecking //
1 Returns the <title> element document.title //
Returns the complete URL of the document document.URL //

JavaScript HTML DOM Elements //

.This page teaches you how to find and access HTML elements in an HTML page //

Finding HTML Elements //
.Often, with JavaScript, you want to manipulate HTML elements //

:To do so, you have to find the elements first. There are several ways to do this //

Finding HTML elements by id //
Finding HTML elements by tag name //

Finding HTML elements by class name //
Finding HTML elements by CSS selectors //
Finding HTML elements by HTML object collections //
Finding HTML Element by Id //
.The easiest way to find an HTML element in the DOM, is by using the element id //

:"This example finds the element with id="intro //

;("const element = document.getElementById("intro //

.(If the element is found, the method will return the element as an object (in element //

.If the element is not found, element will contain null //

Finding HTML Elements by Tag Name //
:This example finds all <p> elements //

;("const element = document.getElementsByTagName("p //

This example finds the element with id="main", and then finds all <p> elements inside //
:""main

;("const x = document.getElementById("main //
;("const y = x.getElementsByTagName("p //

Finding HTML Elements by Class Name //
If you want to find all HTML elements with the same class name, use //
.()getElementsByClassName
."This example returns a list of all elements with class="intro //

;("const x = document.getElementsByClassName("intro //

Finding HTML Elements by CSS Selectors //
If you want to find all HTML elements that match a specified CSS selector (id, class //
.names, types, attributes, values of attributes, etc), use the querySelectorAll() method
."This example returns a list of all <p> elements with class="intro //

;("const x = document.querySelectorAll("p.intro //

Finding HTML Elements by HTML Object Collections //
This example finds the form element with id="frm1", in the forms collection, and displays all //
:element values

;["const x = document.forms["frm1 //
;"" = let text //
} (++for (let i = 0; i < x.length; i //
;"<text += x.elements[i].value + "<br //
{ //

;document.getElementById("demo").innerHTML = text //

:The following HTML objects (and object collections) are also accessible //

document.anchors //
document.body //
document.documentElement //
document.embeds //
document.forms //
document.head //
document.images //
document.links //
document.scripts //
document.title //

JavaScript HTML DOM - Changing HTML //

.The HTML DOM allows JavaScript to change the content of HTML elements //

Changing HTML Content //
The easiest way to modify the content of an HTML element is by using the innerHTML //
.property

:To change the content of an HTML element, use this syntax //

document.getElementById(id).innerHTML = new HTML //
:This example changes the content of a element //

<html> */}
<body>

Hello World!

<script>
;"!document.getElementById("p1").innerHTML = "New text
<script/>

<body/>
{/* <html/>

:Example explained //

"The HTML document above contains a element with id="p1 //
"We use the HTML DOM to get the element with id="p1 //
"!A JavaScript changes the content (innerHTML) of that element to "New text //
:This example changes the content of an <h1> element //

<DOCTYPE html!> //

<html> //
<body> //

<h1 id="id01">Old Heading</h1> //

<script> //
;("const element = document.getElementById("id01 //
;"element.innerHTML = "New Heading //
<script/> //

<body/> //
<html/> //

:Example explained //

"The HTML document above contains an <h1> element with id="id01 //
"We use the HTML DOM to get the element with id="id01 //
"A JavaScript changes the content (innerHTML) of that element to "New Heading //

Changing the Value of an Attribute //
:To change the value of an HTML attribute, use this syntax //

document.getElementById(id).attribute = new value //
:This example changes the value of the src attribute of an <img> element //

<DOCTYPE html!> //
<html> //
<body> //

<"img id="myImage" src="smiley.gif> //

<script> //
;"document.getElementById("myImage").src = "landscape.jpg //
<script/> //

<body/> //
<html/> //

:Example explained //

"The HTML document above contains an <img> element with id="myImage //
"We use the HTML DOM to get the element with id="myImage //
"A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg //
Dynamic HTML content //
:JavaScript can create dynamic HTML content //

(Date : Mon Mar 27 2023 23:19:34 GMT+0330 (Iran Standard Time //

<DOCTYPE html!> //
<html> //
<body> //

<script> //
<document.getElementById("demo").innerHTML = "Date : " + Date(); </script //

<body/> //
<html/> //

()document.write //
:In JavaScript, document.write() can be used to write directly to the HTML output stream //

<DOCTYPE html!> //
<html> //
<body> //

Bla bla bla //

<script> //
;(()document.write(Date //
<script/> //

Bla bla bla //

<body/> //
<html/> //

.Never use document.write() after the document is loaded. It will overwrite the document //

JavaScript Forms //

JavaScript Form Validation //
.HTML form validation can be done by JavaScript //

If a form field (fname) is empty, this function alerts a message, and returns false, to //
:prevent the form from being submitted

} ()function validateForm //
;let x = document.forms["myForm"]["fname"].value //
} ("" == if (x //
;("alert("Name must be filled out //
;return false //
{ //
{ //

:The function can be called when the form is submitted //

form name="myForm" action="/action_page.php" onsubmit="return validateForm()"> //
<"method="post
<"Name: <input type="text" name="fname //
<"input type="submit" value="Submit> //
<form/> //

JavaScript Can Validate Numeric Input //
:JavaScript is often used to validate numeric input //

<"input id="numb> */}

<button type="button" onclick="myFunction()">Submit</button>

<script>
} ()function myFunction
"Get the value of the input field with id="numb //
;let x = document.getElementById("numb").value
If x is Not a Number or less than one or greater than 10 //
;let text
} (if (isNaN(x) || x < 1 || x > 10
;"text = "Input not valid
} else {
;"text = "Input OK
{
;document.getElementById("demo").innerHTML = text
{
{/* <script/>

Automatic HTML Form Validation //
:HTML form validation can be performed automatically by the browser //

If a form field (fname) is empty, the required attribute prevents this form from being //
:submitted

<"form action="/action_page.php" method="post> */}
<input type="text" name="fname" required>
<"input type="submit" value="Submit>
{/* <form/>

.Automatic HTML form validation does not work in Internet Explorer 9 or earlier //

Data Validation //
.Data validation is the process of ensuring that user input is clean, correct, and useful //

:Typical validation tasks are //

?has the user filled in all required fields //
?has the user entered a valid date //
?has the user entered text in a numeric field //
.Most often, the purpose of data validation is to ensure correct user input //

Validation can be defined by many different methods, and deployed in many different //
.ways

Server side validation is performed by a web server, after input has been sent to the //
.server

.Client side validation is performed by a web browser, before input is sent to a web server //

HTML Constraint Validation //
.HTML5 introduced a new HTML validation concept called constraint validation //

:HTML constraint validation is based on //

Constraint validation HTML Input Attributes //
Constraint validation CSS Pseudo Selectors //
Constraint validation DOM Properties and Methods //
Constraint Validation HTML Input Attributes //
Description Attribute //
Specifies that the input element should be disabled disabled //
Specifies the maximum value of an input element max //
Specifies the minimum value of an input element min //
Specifies the value pattern of an input element pattern //
Specifies that the input field requires an element required //
Specifies the type of an input element type //
.For a full list, go to HTML Input Attributes //

Constraint Validation CSS Pseudo Selectors //
Description Selector //
Selects input elements with the "disabled" attribute specified disabled: //
Selects input elements with invalid values invalid: //
Selects input elements with no "required" attribute specified optional: //
Selects input elements with the "required" attribute specified required: //
Selects input elements with valid values valid: //

JavaScript HTML DOM - Changing CSS //

.The HTML DOM allows JavaScript to change the style of HTML elements //

Changing HTML Style //
:To change the style of an HTML element, use this syntax //

document.getElementById(id).style.property = new style //
:The following example changes the style of a <p> element //

<html> //
<body> //

Hello World! //

<script> //
;"document.getElementById("p2").style.color = "blue //
<script/> //

<body/> //
<html/> //

Using Events //
.The HTML DOM allows you to execute code when an event occurs //

:Events are generated by the browser when "things happen" to HTML elements //

An element is clicked on //
The page has loaded //
Input fields are changed //
.You will learn more about events in the next chapter of this tutorial //

This example changes the style of the HTML element with id="id1", when the user clicks a //
:button

<DOCTYPE html!> //
<html> //
<body> //

<h1 id="id1">My Heading 1</h1> //

"button type="button> //
<"'onclick="document.getElementById('id1').style.color = 'red //
<Click Me!</button //

<body/> //
<html/> //

JavaScript HTML DOM Animation //

.Learn to create HTML animations using JavaScript //

A Basic Web Page //
To demonstrate how to create HTML animations with JavaScript, we will use a simple web //
:page

<DOCTYPE html!> //
<html> //
<body> //

<h1>My First JavaScript Animation</h1> //

My animation will go here //

<body/> //
<html/> //

Create an Animation Container //
.All animations should be relative to a container element //

<"div id ="container> //
My animation will go here //
 //

Style the Elements //
."The container element should be created with style = "position: relative //

."The animation element should be created with style = "position: absolute //

<Doctype html!> //
<html> //
<style> //
} container# //
;width: 400px //
;height: 400px //
;position: relative //
;background: yellow //
{ //
} animate# //
;width: 50px //
;height: 50px //
;position: absolute //
;background: red //
{ //
<style/> //
<body> //

<h2>My First JavaScript Animation</h2> //

<"div id="container> //
 //
 //

<body/> //
<html/> //

Animation Code //
.JavaScript animations are done by programming gradual changes in an element's style //

The changes are called by a timer. When the timer interval is small, the animation looks //
.continuous

:The basic code is //

;(id = setInterval(frame, 5 //

} ()function frame //
} (/* if (/* test for finished //
;(clearInterval(id //
} else { //
/* code to change the element style */ //
{ //
{ //

Create the Full Animation Using JavaScript //

} ()function myMove //
;let id = null //
;("const elem = document.getElementById("animate //
;let pos = 0 //
;(clearInterval(id //
;(id = setInterval(frame, 5 //
} ()function frame //
} (if (pos == 350 //
;(clearInterval(id //
} else { //
;++pos //
;'elem.style.top = pos + 'px //
;'elem.style.left = pos + 'px //
{ //
{ //
{ //

JavaScript HTML DOM Events //

Reacting to Events //
A JavaScript can be executed when an event occurs, like when a user clicks on an HTML //
.element

To execute code when a user clicks on an element, add JavaScript code to an HTML event //
:attribute

onclick=JavaScript //
:Examples of HTML events //

When a user clicks the mouse //
When a web page has loaded //
When an image has been loaded //
When the mouse moves over an element //
When an input field is changed //
When an HTML form is submitted //
When a user strokes a key //
:In this example, the content of the <h1> element is changed when a user clicks on it //

<DOCTYPE html!> //
<html> //
<body> //

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1> //

<body/> //
<html/> //

:In this example, a function is called from the event handler //

<DOCTYPE html!> //
<html> //
<body> //

<h1 onclick="changeText(this)">Click on this text!</h1> //

<script> //
} (function changeText(id //
;"!id.innerHTML = "Ooops //
{ //
<script/> //

<body/> //
<html/> //

HTML Event Attributes //
.To assign events to HTML elements you can use event attributes //

:Assign an onclick event to a button element //

<button onclick="displayDate()">Try it</button> //

In the example above, a function named displayDate will be executed when the button is //
.clicked

Assign Events Using the HTML DOM //
:The HTML DOM allows you to assign events to HTML elements using JavaScript //

<script> */}
;document.getElementById("myBtn").onclick = displayDate
{/* <script/>

In the example above, a function named displayDate is assigned to an HTML element with //
."the id="myBtn

.The function will be executed when the button is clicked //

The onload and onunload Events //
.The onload and onunload events are triggered when the user enters or leaves the page //

The onload event can be used to check the visitor's browser type and browser version, and //
.load the proper version of the web page based on the information

.The onload and onunload events can be used to deal with cookies //

{/* <"()body onload="checkCookies> */}

The onchange Event //
.The onchange event is often used in combination with validation of input fields //

Below is an example of how to use the onchange. The upperCase() function will be called //
.when a user changes the content of an input field

{/* <"()input type="text" id="fname" onchange="upperCase> */}

he onmouseover and onmouseout Events //
The onmouseover and onmouseout events can be used to trigger a function when the user //
:mouses over, or out of, an HTML element

"(div onmouseover="mOver(this)" onmouseout="mOut(this> */}
<";style="background-color:#D94A38;width:120px;height:20px;padding:40px
<Mouse Over Me</div

<script>
} (function mOver(obj
"obj.innerHTML = "Thank You
{

} (function mOut(obj
"obj.innerHTML = "Mouse Over Me
{

{/* <script/>

The onmousedown, onmouseup and onclick Events //
The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First //
when a mouse-button is clicked, the onmousedown event is triggered, then, when the
mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is
.completed, the onclick event is triggered

"(div onmousedown="mDown(this)" onmouseup="mUp(this> */}
<";style="background-color:#D94A38;width:90px;height:20px;padding:40px
<Click Me</div

<script>
} (function mDown(obj
;"obj.style.backgroundColor = "#1ec5e5
;"obj.innerHTML = "Release Me
{

} (function mUp(obj
;"obj.style.backgroundColor="#D94A38
;"obj.innerHTML="Thank You
{
{/* <script/>

JavaScript HTML DOM EventListener //

The addEventListener() method //

:Add an event listener that fires when a user clicks a button //

;(document.getElementById("myBtn").addEventListener("click", displayDate //

.The addEventListener() method attaches an event handler to the specified element //

The addEventListener() method attaches an event handler to an element without //
.overwriting existing event handlers

.You can add many event handlers to one element //

.You can add many event handlers of the same type to one element, i.e two "click" events //

You can add event listeners to any DOM object not only HTML elements. i.e the window //
.object

The addEventListener() method makes it easier to control how the event reacts to //
.bubbling

When using the addEventListener() method, the JavaScript is separated from the HTML // markup, for better readability and allows you to add event listeners even when you do not .control the HTML markup

.You can easily remove an event listener by using the removeEventListener() method //

Syntax //
;(element.addEventListener(event, function, useCapture //
The first parameter is the type of the event (like "click" or "mousedown" or any other HTML // (.DOM Event

.The second parameter is the function we want to call when the event occurs //

The third parameter is a boolean value specifying whether to use event bubbling or event // .capturing. This parameter is optional

."Note that you don't use the "on" prefix for the event; use "click" instead of "onclick //

Add an Event Handler to an Element //

:Alert "Hello World!" when the user clicks on an element //

;({ ;("!element.addEventListener("click", function(){ alert("Hello World //

:You can also refer to an external "named" function //

:Alert "Hello World!" when the user clicks on an element //

;(element.addEventListener("click", myFunction //

} ()function myFunction //
;("!alert ("Hello World //
{ //

Add Many Event Handlers to the Same Element //
The addEventListener() method allows you to add many events to the same element, // :without overwriting existing events

;(element.addEventListener("click", myFunction //
;(element.addEventListener("click", mySecondFunction //

:You can add events of different types to the same element //

;(element.addEventListener("mouseover", myFunction //
;(element.addEventListener("click", mySecondFunction //
;(element.addEventListener("mouseout", myThirdFunction //

Add an Event Handler to the window Object //

The addEventListener() method allows you to add event listeners on any HTML DOM //
object such as HTML elements, the HTML document, the window object, or other objects
.that support events, like the xmlHttpRequest object

:Add an event listener that fires when a user resizes the window //

}()window.addEventListener("resize", function //
;document.getElementById("demo").innerHTML = sometext //
;({ //

Passing Parameters //
When passing parameter values, use an "anonymous function" that calls the specified //
:function with the parameters

;({ ;(element.addEventListener("click", function(){ myFunction(p1, p2 //

?Event Bubbling or Event Capturing //
.There are two ways of event propagation in the HTML DOM, bubbling and capturing //

Event propagation is a way of defining the element order when an event occurs. If you //
have a <p> element inside a element, and the user clicks on the <p> element, which
?element's "click" event should be handled first

In bubbling the inner most element's event is handled first and then the outer: the <p> //
.element's click event is handled first, then the element's click event

In capturing the outer most element's event is handled first and then the inner: the //
.element's click event will be handled first, then the <p> element's click event

With the addEventListener() method you can specify the propagation type by using the //
:"useCapture" parameter

;(addEventListener(event, function, useCapture //
The default value is false, which will use the bubbling propagation, when the value is set to //
.true, the event uses the capturing propagation

;(document.getElementById("myP").addEventListener("click", myFunction, true //
;(document.getElementById("myDiv").addEventListener("click", myFunction, true //

The removeEventListener() method //
The removeEventListener() method removes event handlers that have been attached with //
:the addEventListener() method

;(element.removeEventListener("mousemove", myFunction //

JavaScript HTML DOM Navigation //

:From the HTML above you can read //

html> is the root node> //
html> has no parents> //
<html> is the parent of <head> and <body> //
<head> is the first child of <html> //
<body> is the last child of <html> //
:and //

<head> has one child: <title> //
"title> has one child (a text node): "DOM Tutorial> //
<body> has two children: <h1> and <p> //
"h1> has one child: "DOM Lesson one> //
"!p> has one child: "Hello world> //
h1> and <p> are siblings> //

Navigating Between Nodes //
:You can use the following node properties to navigate between nodes with JavaScript //

parentNode //
[childNodes[nodenumber //
firstChild //
lastChild //
nextSibling //
previousSibling //
Child Nodes and Node Values //
.A common error in DOM processing is to expect an element node to contain text //

<title id="demo">DOM Tutorial</title> */}
.The element node <title> (in the example above) does not contain text

."It contains a text node with the value "DOM Tutorial

:The value of the text node can be accessed by the node's innerHTML property

;myTitle = document.getElementById("demo").innerHTML
:Accessing the innerHTML property is the same as accessing the nodeValue of the first child

;myTitle = document.getElementById("demo").firstChild.nodeValue
:Accessing the first child can also be done like this

;myTitle = document.getElementById("demo").childNodes[0].nodeValue
All the (3) following examples retrieves the text of an <h1> element and copies it into a <p>
:element
{/*

<html> */}
<body>

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
;document.getElementById("id01").innerHTML
<script/>

<body/>
{/* <html/>

<html> */}
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
;document.getElementById("id01").firstChild.nodeValue
<script/>

<body/>
{/* <html/>

<html> */}
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
;document.getElementById("id01").firstChild.nodeValue
<script/>

<body/>
{/* <html/>
```

InnerHTML //
.In this tutorial we use the innerHTML property to retrieve the content of an HTML element //

However, learning the other methods above is useful for understanding the tree structure //
.and the navigation of the DOM

DOM Root Nodes //
:There are two special properties that allow access to the full document //

```
document.body - The body of the document //
document.documentElement - The full document //

<html> */}
<body>

<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.body</p>

<p id="demo"></p>

<script>
;document.getElementById("demo").innerHTML = document.body.innerHTML
<script/>

<body/>
<html/>

<html>
<body>

<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.documentElement</p>

<p id="demo"></p>

<script>
;document.getElementById("demo").innerHTML = document.documentElement.innerHTML
<script/>

<body/>
{/* <html/>

The nodeName Property //
.The nodeName property specifies the name of a node //

nodeName is read-only //
nodeName of an element node is the same as the tag name //
nodeName of an attribute node is the attribute name //
nodeName of a text node is always #text //
nodeName of the document node is always #document //

<h1 id="id01">My First Page</h1> */}
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
;document.getElementById("id01").nodeName
```

{/* <script/>

.Note: nodeName always contains the uppercase tag name of an HTML element //

The nodeValue Property //
.The nodeValue property specifies the value of a node //

nodeValue for element nodes is null //
nodeValue for text nodes is the text itself //
nodeValue for attribute nodes is the attribute value //
The nodeType Property //
.The nodeType property is read only. It returns the type of a node //

<h1 id="id01">My First Page</h1> */}

<script>
document.getElementById("id02").innerHTML =
;document.getElementById("id01").nodeType
{/* <script/>

:The most important nodeType properties are //

Example TypeNode //
<<h1 class="heading">W3Schools</h1 1 ELEMENT_NODE //
(class = "heading" (deprecated 2ATTRIBUTE_NODE //
W3Schools 3 TEXT_NODE //
<-- <!-- This is a comment 8 COMMENT_NODE //
(<The HTML document itself (the parent of <html 9 DOCUMENT_NODE //
<<!Doctype html 10 DOCUMENT_TYPE_NODE //
.Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM //

(JavaScript HTML DOM Elements (Nodes //

(Adding and Removing Nodes (HTML Elements //

(Creating New HTML Elements (Nodes //
To add a new element to the HTML DOM, you must create the element (element node) //
.first, and then append it to an existing element

<"div id="div1> //
This is a paragraph. //
This is another paragraph. //
 //

<script> //
;("const para = document.createElement("p //
;(".const node = document.createTextNode("This is new //

;(para.appendChild(node //

;("const element = document.getElementById("div1 //
;(element.appendChild(para //
<script/> //

Example Explained //
:This code creates a new element //

;("const para = document.createElement("p //
To add text to the element, you must create a text node first. This code creates a text //
:node

;(".const node = document.createTextNode("This is a new paragraph //
:Then you must append the text node to the element //

;(para.appendChild(node //
.Finally you must append the new element to an existing element //

:This code finds an existing element //

;("const element = document.getElementById("div1 //
:This code appends the new element to the existing element //

;(element.appendChild(para //

()Creating new HTML Elements - insertBefore //
The appendChild() method in the previous example, appended the new element as the last //
.child of the parent

:If you don't want that you can use the insertBefore() method //

<"div id="div1> */}
This is a paragraph.
This is another paragraph.

<script>
;("const para = document.createElement("p
;(".const node = document.createTextNode("This is new
;(para.appendChild(node

;("const element = document.getElementById("div1
;("const child = document.getElementById("p1
;(element.insertBefore(para, child
{/* <script/>

Removing Existing HTML Elements //

:To remove an HTML element, use the remove() method //

 //
This is a paragraph. //
This is another paragraph. //
 //

<script> //
;()const elmnt = document.getElementById("p1"); elmnt.remove //
<script/> //

:(The HTML document contains a element with two child nodes (two elements //

 //
This is a paragraph. //
This is another paragraph. //
 //
:Find the element you want to remove //

;("const elmnt = document.getElementById("p1 //
:Then execute the remove() method on that element //

;()elmnt.remove //
The remove() method does not work in older browsers, see the example below on how to //
.use removeChild() instead

Removing a Child Node //
For browsers that does not support the remove() method, you have to find the parent node //
:to remove an element

<"div id="div1> */}
This is a paragraph.
This is another paragraph.

<script>
;("const parent = document.getElementById("div1
;("const child = document.getElementById("p1
;(parent.removeChild(child
{/* <script/>

:(This HTML document contains a element with two child nodes (two elements //

<"div id="div1> //
This is a paragraph. //
This is another paragraph. //
 //
:"Find the element with id="div1 //

;("const parent = document.getElementById("div1 //
:"Find the element with id="p1 //

;("const child = document.getElementById("p1 //
:Remove the child from the parent //

;(parent.removeChild(child //
Here is a common workaround: Find the child you want to remove, and use its parentNode //
:property to find the parent

;("const child = document.getElementById("p1 //
;(child.parentNode.removeChild(child //
Replacing HTML Elements //
:To replace an element to the HTML DOM, use the replaceChild() method //

<"div id="div1> */}
This is a paragraph.
This is another paragraph.

<script>
;("const para = document.createElement("p
;(".const node = document.createTextNode("This is new
;(para.appendChild(node

;("const parent = document.getElementById("div1
;("const child = document.getElementById("p1
;(parent.replaceChild(para, child
{/* <script/>

JavaScript HTML DOM Collections //

The HTMLCollection Object //
.The getElementsByTagName() method returns an HTMLCollection object //

.An HTMLCollection object is an array-like list (collection) of HTML elements //

:The following code selects all elements in a document //

;("const myCollection = document.getElementsByTagName("p //
.The elements in the collection can be accessed by an index number //

:To access the second element you can write //

[myCollection[1 //

.Note: The index starts at 0 //

HTML HTMLCollection Length //
:The length property defines the number of elements in an HTMLCollection //

myCollection.length //

:The length property is useful when you want to loop through the elements in a collection //

:Change the text color of all <p> elements //

;("const myCollection = document.getElementsByTagName("p //
} (++for (let i = 0; i < myCollection.length; i //
;"myCollection[i].style.color = "red //
{ //

!An HTMLCollection is NOT an array //

.An HTMLCollection may look like an array, but it is not //

.(You can loop through the list and refer to the elements with a number (just like an array //

However, you cannot use array methods like valueOf(), pop(), push(), or join() on an //
.HTMLCollection

JavaScript HTML DOM Node Lists //

The HTML DOM NodeList Object //
.A NodeList object is a list (collection) of nodes extracted from a document //

.A NodeList object is almost the same as an HTMLCollection object //

Some (older) browsers return a NodeList object instead of an HTMLCollection for methods //
.()like getElementsByClassName

.All browsers return a NodeList object for the property childNodes //

.()Most browsers return a NodeList object for the method querySelectorAll //

:The following code selects all <p> nodes in a document //

;("const myNodeList = document.querySelectorAll("p //
.The elements in the NodeList can be accessed by an index number //

:To access the second <p> node you can write //

[myNodeList[1 //

.Note: The index starts at 0 //

HTML DOM Node List Length //
:The length property defines the number of nodes in a node list //

myNodelist.length //

:The length property is useful when you want to loop through the nodes in a node list //

:Change the color of all <p> elements in a node list //

;("const myNodelist = document.querySelectorAll("p //
} (++for (let i = 0; i < myNodelist.length; i //
;"myNodelist[i].style.color = "red //
{ //

The Difference Between an HTMLCollection and a NodeList //
.A NodeList and an HTMLcollection is very much the same thing //

Both are array-like collections (lists) of nodes (elements) extracted from a document. The //
.nodes can be accessed by index numbers. The index starts at 0

.(Both have a length property that returns the number of elements in the list (collection //

.An HTMLCollection is a collection of document elements //

A NodeList is a collection of document nodes (element nodes, attribute nodes, and text //
.(nodes

.HTMLCollection items can be accessed by their name, id, or index number //

.NodeList items can only be accessed by their index number //

An HTMLCollection is always a live collection. Example: If you add a <li> element to a list //
.in the DOM, the list in the HTMLCollection will also change

A NodeList is most often a static collection. Example: If you add a <li> element to a list in //
.the DOM, the list in NodeList will not change

The getElementsByClassName() and getElementsByTagName() methods return a live //
.HTMLCollection

.The querySelectorAll() method returns a static NodeList //

.The childNodes property returns a live NodeList //

!Not an Array //
.A NodeList may look like an array, but it is not //

.You can loop through a NodeList and refer to its nodes by index //

.But, you cannot use Array methods like push(), pop(), or join() on a NodeList //

JavaScript Window - The Browser Object Model //

.The Browser Object Model (BOM) allows JavaScript to "talk to" the browser //

(The Browser Object Model (BOM //
.(There are no official standards for the Browser Object Model (BOM //

Since modern browsers have implemented (almost) the same methods and properties for //
.JavaScript interactivity, it is often referred to, as methods and properties of the BOM

The Window Object //
.The window object is supported by all browsers. It represents the browser's window //

All global JavaScript objects, functions, and variables automatically become members of //
.the window object

.Global variables are properties of the window object //

.Global functions are methods of the window object //

:Even the document object (of the HTML DOM) is a property of the window object //

;("window.document.getElementById("header //
:is the same as //

;("document.getElementById("header //
Window Size //
.Two properties can be used to determine the size of the browser window //

:Both properties return the sizes in pixels //

(window.innerHeight - the inner height of the browser window (in pixels //
(window.innerWidth - the inner width of the browser window (in pixels //
.The browser window (the browser viewport) is NOT including toolbars and scrollbars //

;let w = window.innerWidth //
;let h = window.innerHeight //

Other Window Methods //
:Some other methods //

window.open() - open a new window //
window.close() - close the current window //
window.moveTo() - move the current window //

window.resizeTo() - resize the current window //

JavaScript Window Screen //

.The window.screen object contains information about the user's screen //

Window Screen //
.The window.screen object can be written without the window prefix //

:Properties //

screen.width //
screen.height //
screen.availWidth //
screen.availHeight //
screen.colorDepth //
screen.pixelDepth //
Window Screen Width //
.The screen.width property returns the width of the visitor's screen in pixels //

:Display the width of the screen in pixels //

= document.getElementById("demo").innerHTML //
;Screen Width: " + screen.width" //
:Result will be //

Screen Width: 1280 //

Window Screen Height //
.The screen.height property returns the height of the visitor's screen in pixels //

:Display the height of the screen in pixels //

= document.getElementById("demo").innerHTML //
;Screen Height: " + screen.height" //
:Result will be //

Screen Height: 800 //

Window Screen Available Width //
The screen.availWidth property returns the width of the visitor's screen, in pixels, minus //
.interface features like the Windows Taskbar

:Display the available width of the screen in pixels //

= document.getElementById("demo").innerHTML //
;Available Screen Width: " + screen.availWidth" //
:Result will be //

Available Screen Width: 1280 //

Window Screen Available Height //
The screen.availHeight property returns the height of the visitor's screen, in pixels, minus //
.interface features like the Windows Taskbar

:Display the available height of the screen in pixels //

= document.getElementById("demo").innerHTML //
;Available Screen Height: " + screen.availHeight" //
:Result will be //

Available Screen Height: 770 //

Window Screen Color Depth //
.The screen.colorDepth property returns the number of bits used to display one color //

:All modern computers use 24 bit or 32 bit hardware for color resolution //

"bits = 16,777,216 different "True Colors 24 //
"bits = 4,294,967,296 different "Deep Colors 32 //
.Older computers used 16 bits: 65,536 different "High Colors" resolution //

."Very old computers, and old cell phones used 8 bits: 256 different "VGA colors //

:Display the color depth of the screen in bits //

= document.getElementById("demo").innerHTML //
;Screen Color Depth: " + screen.colorDepth" //
:Result will be //

Screen Color Depth: 24 //

The #rrggbb (rgb) values used in HTML represents "True Colors" (16,777,216 different //
(colors

Window Screen Pixel Depth //
.The screen.pixelDepth property returns the pixel depth of the screen //

:Display the pixel depth of the screen in bits //

= document.getElementById("demo").innerHTML //
;Screen Pixel Depth: " + screen.pixelDepth" //
:Result will be //

Screen Pixel Depth: 24 //

.For modern computers, Color Depth and Pixel Depth are equal //

JavaScript Window Location //

The window.location object can be used to get the current page address (URL) and to //
.redirect the browser to a new page

Window Location //
.The window.location object can be written without the window prefix //

:Some examples //

window.location.href returns the href (URL) of the current page //
window.location.hostname returns the domain name of the web host //
window.location.pathname returns the path and filename of the current page //
(:window.location.protocol returns the web protocol used (http: or https //
window.location.assign() loads a new document //
Window Location Href //
.The window.location.href property returns the URL of the current page //

:Display the href (URL) of the current page //

= document.getElementById("demo").innerHTML //
;Page location is " + window.location.href" //
:Result is //

Page location is https://www.w3schools.com/js/js_window_location.asp //

Window Location Hostname //
The window.location.hostname property returns the name of the internet host (of the //
.(current page

:Display the name of the host //

= document.getElementById("demo").innerHTML //
;Page hostname is " + window.location.hostname" //
:Result is //

Page hostname is www.w3schools.com //

Window Location Pathname //
.The window.location.pathname property returns the pathname of the current page //

:Display the path name of the current URL //

= document.getElementById("demo").innerHTML //
;Page path is " + window.location.pathname" //
:Result is //

Page path is /js/js_window_location.asp //

Window Location Protocol //
.The window.location.protocol property returns the web protocol of the page //

:Display the web protocol //

= document.getElementById("demo").innerHTML //
;Page protocol is " + window.location.protocol" //
:Result is //

:Page protocol is https //

Window Location Port //
The window.location.port property returns the number of the internet host port (of the //
.(current page

:Display the name of the host //

= document.getElementById("demo").innerHTML //
;Port number is " + window.location.port" //
:Result is //

Port number is //

(Most browsers will not display default port numbers (80 for http and 443 for https //

Window Location Assign //
.The window.location.assign() method loads a new document //

:Load a new document //

<html> //
<head> //
<script> //
} ()function newDoc //
("window.location.assign("https://www.w3schools.com //
{ //
<script/> //
<head/> //
<body> //

<"()input type="button" value="Load new document" onclick="newDoc> //

<body/> //
<html/> //

JavaScript Window History //

.The window.history object contains the browsers history //

Window History //
.The window.history object can be written without the window prefix //

To protect the privacy of the users, there are limitations to how JavaScript can access this //
.object

:Some methods //

history.back() - same as clicking back in the browser //
history.forward() - same as clicking forward in the browser //
Window History Back //
.The history.back() method loads the previous URL in the history list //

.This is the same as clicking the Back button in the browser //

:Create a back button on a page //

<html> //
<head> //
<script> //
} ()function goBack //
()window.history.back //
{ //
<script/> //
<head/> //
<body> //

<"()input type="button" value="Back" onclick="goBack> //

<body/> //
<html/> //
:The output of the code above will be //

Back //

Window History Forward //
.The history.forward() method loads the next URL in the history list //

.This is the same as clicking the Forward button in the browser //

:Create a forward button on a page //

<html> //
<head> //

<script> //
} ()function goForward //
()window.history.forward   //
{ //
<script/> //
<head/> //
<body> //

<"()input type="button" value="Forward" onclick="goForward> //

<body/> //
<html/> //
:The output of the code above will be //

Forward //

JavaScript Window Navigator //

.The window.navigator object contains information about the visitor's browser //

Window Navigator //
.The window.navigator object can be written without the window prefix //

:Some examples //

navigator.cookieEnabled //
navigator.appCodeName //
navigator.platform //
Browser Cookies //
:The cookieEnabled property returns true if cookies are enabled, otherwise false //

 */}

<script>
= document.getElementById("demo").innerHTML
;cookiesEnabled is " + navigator.cookieEnabled"
{/* <script/>

Browser Application Name //
:The appName property returns the application name of the browser //

 */}

<script>
= document.getElementById("demo").innerHTML
;navigator.appName is " + navigator.appName"
{/* <script/>

Warning //
.This property is removed (deprecated) in the latest web standard //

.Most browsers (IE11, Chrome, Firefox, Safari) returns Netscape as appName //

Browser Application Code Name //
:The appCodeName property returns the application code name of the browser //

 */}

<script>
= document.getElementById("demo").innerHTML
;navigator.appCodeName is " + navigator.appCodeName"
{/* <script/>

Warning //
.This property is removed (deprecated) in the latest web standard //

.Most browsers (IE11, Chrome, Firefox, Safari, Opera) returns Mozilla as appCodeName //

The Browser Engine //
:The product property returns the product name of the browser engine //

 */}

<script>
= document.getElementById("demo").innerHTML
;navigator.product is " + navigator.product"
{/* <script/>

Warning //
.This property is removed (deprecated) in the latest web standard //

.Most browsers returns Gecko as product //

The Browser Version //
:The appVersion property returns version information about the browser //

 */}

<script>
;document.getElementById("demo").innerHTML = navigator.appVersion
{/* <script/>

The Browser Agent //
:The userAgent property returns the user-agent header sent by the browser to the server //

 */}

<script>
;document.getElementById("demo").innerHTML = navigator.userAgent
{/* <script/>

Warning //
.The information from the navigator object can often be misleading //

:The navigator object should not be used to detect browser versions because //

Different browsers can use the same name //
The navigator data can be changed by the browser owner //
Some browsers misidentify themselves to bypass site tests //
Browsers cannot report new operating systems, released later than the browser //
The Browser Platform //
:(The platform property returns the browser platform (operating system //

 */}

<script>
;document.getElementById("demo").innerHTML = navigator.platform
{/* <script/>

The Browser Language //
:The language property returns the browser's language //

 */}

<script>
;document.getElementById("demo").innerHTML = navigator.language
{/* <script/>

?Is The Browser Online //
:The onLine property returns true if the browser is online //

 */}

<script>
;document.getElementById("demo").innerHTML = navigator.onLine
{/* <script/>

?Is Java Enabled //
:The javaEnabled() method returns true if Java is enabled //

 */}

<script>

;()document.getElementById("demo").innerHTML = navigator.javaEnabled
{/* <script/>

JavaScript Popup Boxes //

.JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box //

Alert Box //
.An alert box is often used if you want to make sure information comes through to the user //

.When an alert box pops up, the user will have to click "OK" to proceed //

Syntax //
;("window.alert("sometext //
.The window.alert() method can be written without the window prefix //

;("!alert("I am an alert box //

Confirm Box //
.A confirm box is often used if you want the user to verify or accept something //

.When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed //

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns //
.false

Syntax //
;("window.confirm("sometext //
.The window.confirm() method can be written without the window prefix //

} (("!if (confirm("Press a button //
;"!txt = "You pressed OK //
} else { //
;"!txt = "You pressed Cancel //
{ //

Prompt Box //
.A prompt box is often used if you want the user to input a value before entering a page //

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed //
.after entering an input value

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box //
.returns null

Syntax //
;("window.prompt("sometext","defaultText //
.The window.prompt() method can be written without the window prefix //

;("let person = prompt("Please enter your name", "Harry Potter //
;let text //
} ("" == if (person == null || person //
;".text = "User cancelled the prompt   //
} else { //
;"?text = "Hello " + person + "! How are you today   //
{ //

Line Breaks //
.To display line breaks inside a popup box, use a back-slash followed by the character n //

;("?alert("Hello\nHow are you //

JavaScript Timing Events //

Timing Events //
.The window object allows execution of code at specified time intervals //

.These time intervals are called timing events //

:The two key methods to use with JavaScript are //

(setTimeout(function, milliseconds //
.Executes a function, after waiting a specified number of milliseconds //

(setInterval(function, milliseconds //
.Same as setTimeout(), but repeats the execution of the function continuously //
.The setTimeout() and setInterval() are both methods of the HTML DOM Window object //

The setTimeout() Method //
;(window.setTimeout(function, milliseconds //
.The window.setTimeout() method can be written without the window prefix //

.The first parameter is a function to be executed //

.The second parameter indicates the number of milliseconds before execution //

:"Click a button. Wait 3 seconds, and the page will alert "Hello //

<button onclick="setTimeout(myFunction, 3000)">Try it</button> //

<script> //
} ()function myFunction //
;('alert('Hello   //
{ //
<script/> //

?How to Stop the Execution //
.()The clearTimeout() method stops the execution of the function specified in setTimeout //

(window.clearTimeout(timeoutVariable //
.The window.clearTimeout() method can be written without the window prefix //

:()The clearTimeout() method uses the variable returned from setTimeout //

;(myVar = setTimeout(function, milliseconds //
;(clearTimeout(myVar //
If the function has not already been executed, you can stop the execution by calling the //
:clearTimeout() method

:Same example as above, but with an added "Stop" button //

<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button> //

button onclick="clearTimeout(myVar)">Stop it</button> //

The setInterval() Method //
.The setInterval() method repeats a given function at every given time-interval //

;(window.setInterval(function, milliseconds //
.The window.setInterval() method can be written without the window prefix //

.The first parameter is the function to be executed //

.The second parameter indicates the length of the time-interval between each execution //

This example executes a function called "myTimer" once every second (like a digital //
.(watch

:Display the current time //

;(setInterval(myTimer, 1000 //

} ()function myTimer //
;()const d = new Date //
;()document.getElementById("demo").innerHTML = d.toLocaleTimeString //
{ //

.There are 1000 milliseconds in one second //

?How to Stop the Execution //
The clearInterval() method stops the executions of the function specified in the setInterval() //
.method

(window.clearInterval(timerVariable //

.The window.clearInterval() method can be written without the window prefix //

:()The clearInterval() method uses the variable returned from setInterval //

;(let myVar = setInterval(function, milliseconds //
;(clearInterval(myVar //

:Same example as above, but we have added a "Stop time" button //

 //

<button onclick="clearInterval(myVar)">Stop time</button> //

<script> //
;(let myVar = setInterval(myTimer, 1000 //
} ()function myTimer //
;()const d = new Date //
;()document.getElementById("demo").innerHTML = d.toLocaleTimeString //
{ //
<script/> //

JavaScript Cookies //

?What are Cookies //
.Cookies are data, stored in small text files, on your computer //

When a web server has sent a web page to a browser, the connection is shut down, and //
.the server forgets everything about the user

Cookies were invented to solve the problem "how to remember information about the //
:"user

.When a user visits a web page, his/her name can be stored in a cookie //
.Next time the user visits the page, the cookie "remembers" his/her name //
:Cookies are saved in name-value pairs like //

username = John Doe //
When a browser requests a web page from a server, cookies belonging to the page are //
added to the request. This way the server gets the necessary data to "remember"
.information about users

.None of the examples below will work if your browser has local cookies support turned off //

Create a Cookie with JavaScript //
.JavaScript can create, read, and delete cookies with the document.cookie property //

:With JavaScript, a cookie can be created like this //

;"document.cookie = "username=John Doe //
You can also add an expiry date (in UTC time). By default, the cookie is deleted when the //
:browser is closed

;"document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC //
With a path parameter, you can tell the browser what path the cookie belongs to. By //
.default, the cookie belongs to the current page

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; //
;"/=path

Read a Cookie with JavaScript //
:With JavaScript, cookies can be read like this //

;let x = document.cookie //
document.cookie will return all cookies in one string much like: cookie1=value; //
;cookie2=value; cookie3=value

Change a Cookie with JavaScript //
:With JavaScript, you can change a cookie the same way as you create it //

document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; //
;"/=path
.The old cookie is overwritten //

Delete a Cookie with JavaScript //
.Deleting a cookie is very simple //

.You don't have to specify a cookie value when you delete a cookie //

:Just set the expires parameter to a past date //

;";/=document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path //
.You should define the cookie path to ensure that you delete the right cookie //

.Some browsers will not let you delete a cookie if you don't specify the path //

The Cookie String //
.The document.cookie property looks like a normal text string. But it is not //

Even if you write a whole cookie string to document.cookie, when you read it out again, //
.you can only see the name-value pair of it

If you set a new cookie, older cookies are not overwritten. The new cookie is added to //
:document.cookie, so if you read document.cookie again you will get something like

;cookie1 = value; cookie2 = value //

Display All Cookies Create Cookie 1 Create Cookie 2 Delete Cookie 1 Delete Cookie 2 //

If you want to find the value of one specified cookie, you must write a JavaScript function //
.that searches for the cookie value in the cookie string

JavaScript Cookie Example //
.In the example to follow, we will create a cookie that stores the name of a visitor //

The first time a visitor arrives to the web page, he/she will be asked to fill in his/her name. //
.The name is then stored in a cookie

.The next time the visitor arrives at the same page, he/she will get a welcome message //

:For the example we will create 3 JavaScript functions //

A function to set a cookie value //
A function to get a cookie value //
A function to check a cookie value //
A Function to Set a Cookie //
:First, we create a function that stores the name of the visitor in a cookie variable //

} (function setCookie(cname, cvalue, exdays //
;()const d = new Date   //
;((d.setTime(d.getTime() + (exdays*24*60*60*1000   //
;()let expires = "expires="+ d.toUTCString   //
;"/=document.cookie = cname + "=" + cvalue + ";" + expires + ";path   //
{ //
:Example explained //

The parameters of the function above are the name of the cookie (cname), the value of the //
.(cookie (cvalue), and the number of days until the cookie should expire (exdays

The function sets a cookie by adding together the cookiename, the cookie value, and the //
.expires string

A Function to Get a Cookie //
:Then, we create a function that returns the value of a specified cookie //

} (function getCookie(cname //
;"=" + let name = cname   //
;(let decodedCookie = decodeURIComponent(document.cookie   //
;(';')let ca = decodedCookie.split   //
} (++for(let i = 0; i <ca.length; i   //
;[let c = ca[i   //
} (' ' == (while (c.charAt(0   //
;(c = c.substring(1   //
{   //
} (if (c.indexOf(name) == 0   //

;(return c.substring(name.length, c.length   //
{   //
{   //
;"" return   //
{ //

:Function explained //

.(Take the cookiename as parameter (cname //

.("=" + Create a variable (name) with the text to search for (cname //

'$' .Decode the cookie string, to handle cookies with special characters, e.g //

Split document.cookie on semicolons into an array called ca (ca = //
.((';')decodedCookie.split

.([Loop through the ca array (i = 0; i < ca.length; i++), and read out each value c = ca[i //

If the cookie is found (c.indexOf(name) == 0), return the value of the cookie //
.((c.substring(name.length, c.length

."" If the cookie is not found, return //

A Function to Check a Cookie //
.Last, we create the function that checks if a cookie is set //

.If the cookie is set it will display a greeting //

If the cookie is not set, it will display a prompt box, asking for the name of the user, and //
:stores the username cookie for 365 days, by calling the setCookie function

} ()function checkCookie //
;("let username = getCookie("username   //
} ("" =! if (username   //
;(alert("Welcome again " + username   //
} else {   //
;("" ,":username = prompt("Please enter your name   //
} (if (username != "" && username != null   //
;(setCookie("username", username, 365   //
{   //
{   //
{ //

All Together Now //

} (function setCookie(cname, cvalue, exdays //

```
;()const d = new Date   //
;((d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000   //
;()let expires = "expires="+d.toUTCString   //
;"/=document.cookie = cname + "=" + cvalue + ";" + expires + ";path   //
{ //

} (function getCookie(cname //
;"=" + let name = cname   //
;(';')let ca = document.cookie.split   //
} (++for(let i = 0; i < ca.length; i   //
;[let c = ca[i     //
} (' ' == (while (c.charAt(0     //
;(c = c.substring(1       //
{     //
} (if (c.indexOf(name) == 0     //
;(return c.substring(name.length, c.length       //
{     //
{   //
;"" return   //
{ //

} ()function checkCookie //
;("let user = getCookie("username   //
} ("" =! if (user   //
;(alert("Welcome again " + user     //
} else {   //
;("" ,":user = prompt("Please enter your name     //
} (if (user != "" && user != null     //
;(setCookie("username", user, 365       //
{     //
{   //
{ //
```

.The example above runs the checkCookie() function when the page loads //

Web APIs - Introduction //

.A Web API is a developer's dream //

It can extend the functionality of the browser //
It can greatly simplify complex functions //
It can provide easy syntax to complex code //
?What is Web API //
.API stands for Application Programming Interface //

.A Web API is an application programming interface for the Web //

.A Browser API can extend the functionality of a web browser //

.A Server API can extend the functionality of a web server //

Browser APIs //
All browsers have a set of built-in Web APIs to support complex operations, and to help //
.accessing data

For example, the Geolocation API can return the coordinates of where the browser is //
.located

:Get the latitude and longitude of the user's position //

;("const myElement = document.getElementById("demo //

} ()function getLocation //
} (if (navigator.geolocation //
;(navigator.geolocation.getCurrentPosition(showPosition //
} else { //
;".myElement.innerHTML = "Geolocation is not supported by this browser //
{ //
{ //

} (function showPosition(position //
+ myElement.innerHTML = "Latitude: " + position.coords.latitude //
;br>Longitude: " + position.coords.longitude>" //
{ //

Third Party APIs //
.Third party APIs are not built into your browser //

.To use these APIs, you will have to download the code from the Web //

:Examples //

.YouTube API - Allows you to display videos on a web site //
.Twitter API - Allows you to display Tweets on a web site //
.Facebook API - Allows you to display Facebook info on a web site //

JavaScript Validation API //

Constraint Validation DOM Methods //
Description Property //
.Returns true if an input element contains valid data checkValidity() //
.Sets the validationMessage property of an input element setCustomValidity() //
:If an input field contains invalid data, display a message //

The checkValidity() Method //
<input id="id1" type="number" min="100" max="300" required> //

<button onclick="myFunction()">OK</button> //

 //

<script> //
} ()function myFunction //
;("const inpObj = document.getElementById("id1   //
} (()if (!inpObj.checkValidity   //
;document.getElementById("demo").innerHTML = inpObj.validationMessage   //
{   //
{ //
<script/> //

Constraint Validation DOM Properties //
Description   Property //
.Contains boolean properties related to the validity of an input element   validity //
.Contains the message a browser will display when the validity is false   validationMessage //
.Indicates if an input element will be validated   willValidate //

Validity Properties //
The validity property of an input element contains a number of properties related to the //
:validity of data

Description   Property //
.Set to true, if a custom validity message is set customError //
.Set to true, if an element's value does not match its pattern attribute   patternMismatch //
.Set to true, if an element's value is greater than its max attribute   rangeOverflow //
.Set to true, if an element's value is less than its min attribute   rangeUnderflow //
.Set to true, if an element's value is invalid per its step attribute   stepMismatch //
.Set to true, if an element's value exceeds its maxLength attribute   tooLong //
.Set to true, if an element's value is invalid per its type attribute   typeMismatch //
.Set to true, if an element (with a required attribute) has no value   valueMissing //
.Set to true, if an element's value is valid valid //

If the number in an input field is greater than 100 (the input's max attribute), display a //
:message

The rangeOverflow Property //
<"input id="id1" type="number" max="100> //
<button onclick="myFunction()">OK</button> //

 //

<script> //
} ()function myFunction //
;"let text = "Value OK   //
} (if (document.getElementById("id1").validity.rangeOverflow   //
;"text = "Value too large   //

{   //
{ //
<script/> //

If the number in an input field is less than 100 (the input's min attribute), display a //
:message

The rangeUnderflow Property //
<"input id="id1" type="number" min="100> //
<button onclick="myFunction()">OK</button> //

 //

<script> //
} ()function myFunction //
;"let text = "Value OK   //
} (if (document.getElementById("id1").validity.rangeUnderflow   //
;"text = "Value too small   //
{   //
{ //
<script/> //

Web History API //

.The Web History API provides easy methods to access the windows.history object //

.The window.history object contains the URLs (Web Sites) visited by the user //

The History back() Method //
.The back() method loads the previous URL in the windows.history list //

.It is the same as clicking the "back arrow" in your browser //

<button onclick="myFunction()">Go Back</button> */}

<script>
} ()function myFunction
;()window.history.back
{
{/* <script/>

The History go() Method //
:The go() method loads a specific URL from the history list //

<button onclick="myFunction()">Go Back 2 Pages</button> */}

<script>
} ()function myFunction

;(window.history.go(-2
{
{/* <script/>

History Object Properties //
Description Property //
Returns the number of URLs in the history list length //
History Object Methods //
Description Method //
Loads the previous URL in the history list back() //
Loads the next URL in the history list forward() //
Loads a specific URL from the history list go() //

Web Storage API //

The Web Storage API is a simple syntax for storing and retrieving data in the browser. It is //
:very easy to use

;("localStorage.setItem("name", "John Doe //
;("localStorage.getItem("name //

The localStorage Object //
The localStorage object provides access to a local storage for a particular Web Site. It //
.allows you to store, read, add, modify, and delete data items for that domain

The data is stored with no expiration date, and will not be deleted when the browser is //
.closed

.The data will be available for days, weeks, and years //

The setItem() Method //
.The localStorage.setItem() method stores a data item in a storage //

:It takes a name and a value as parameters //

;("localStorage.setItem("name", "John Doe //

The getItem() Method //
.The localStorage.getItem() method retrieves a data item from the storage //

:It takes a name as parameter //

;("localStorage.getItem("name //

The sessionStorage Object //
.The sessionStorage object is identical to the localStorage object //

.The difference is that the sessionStorage object stores data for one session //

.The data is deleted when the browser is closed //

;("sessionStorage.getItem("name //

The setItem() Method //
.The sessionStorage.setItem() method stores a data item in a storage //

:It takes a name and a value as parameters //

;("sessionStorage.setItem("name", "John Doe //

The getItem() Method //
.The sessionStorage.getItem() method retrieves a data item from the storage //

:It takes a name as parameter //

;("sessionStorage.getItem("name //

Storage Object Properties and Methods //
Description Property/Method //
Returns the name of the nth key in the storage key(n) //
Returns the number of data items stored in the Storage object length //
Returns the value of the specified key name getItem(keyname) //
Adds a key to the storage, or updates a key value (if it already setItem(keyname, value) //
(exists
Removes that key from the storage removeItem(keyname) //
Empty all key out of the storage clear() //
Related Pages for Web Storage API //
Description Property //
Allows to save key/value pairs in a web browser. Stores the window.localStorage //
data with no expiration date
Allows to save key/value pairs in a web browser. Stores the window.sessionStorage //
data for one session

Web Workers API //

A web worker is a JavaScript running in the background, without affecting the performance //
.of the page

?What is a Web Worker //
When executing scripts in an HTML page, the page becomes unresponsive until the script //
.is finished

A web worker is a JavaScript that runs in the background, independently of other scripts, //
without affecting the performance of the page. You can continue to do whatever you want:
.clicking, selecting things, etc., while the web worker runs in the background

Web Workers Example //
:The example below creates a simple web worker that count numbers in the background //

<DOCTYPE html!> //
<html> //
<body> //

<h2>JavaScript Web Workers API</h2> //
Count numbers: <output id="result"></output> //
<button onclick="startWorker()">Start Worker</button> //
<button onclick="stopWorker()">Stop Worker</button> //

<script> //
;let w //

} ()function startWorker //
} ("if(typeof(w) == "undefined //
;("w = new Worker("demo_workers.js //
{ //
} (w.onmessage = function(event //
;document.getElementById("result").innerHTML = event.data //
;{ //
{ //

} ()function stopWorker //
;()w.terminate //
;w = undefined //
{ //
<script/> //

<body/> //
<html/> //

Check Web Worker Support //
:Before creating a web worker, check whether the user's browser supports it //

} ("if (typeof(Worker) !== "undefined //
!Yes! Web worker support // //
.....Some code // //
} else { //
..Sorry! No Web Worker support // //
{ //

Create a Web Worker File //
.Now, let's create our web worker in an external JavaScript //

:Here, we create a script that counts. The script is stored in the "demo_workers.js" file //

```
;let i = 0 //

} ()function timedCount //
;++ i   //
;(postMessage(i   //
;(setTimeout("timedCount()",500   //
{ //

;()timedCount //
```
The important part of the code above is the postMessage() method - which is used to post //
.a message back to the HTML page

Note: Normally web workers are not used for such simple scripts, but for more CPU //
.intensive tasks

Create a Web Worker Object //
.Now that we have the web worker file, we need to call it from an HTML page //

The following lines checks if the worker already exists, if not - it creates a new web worker //
:"object and runs the code in "demo_workers.js

```
} ("if (typeof(w) == "undefined //
;("w = new Worker("demo_workers.js   //
{ //
```
.Then we can send and receive messages from the web worker //

.Add an "onmessage" event listener to the web worker //

```
}(w.onmessage = function(event //
;document.getElementById("result").innerHTML = event.data   //
;{ //
```
When the web worker posts a message, the code within the event listener is executed. //
.The data from the web worker is stored in event.data

Terminate a Web Worker //
When a web worker object is created, it will continue to listen for messages (even after the //
.external script is finished) until it is terminated

To terminate a web worker, and free browser/computer resources, use the terminate() //
:method

```
;()w.terminate //
```
Reuse the Web Worker //
If you set the worker variable to undefined, after it has been terminated, you can reuse the //
:code

```
;w = undefined //
```
Full Web Worker Example Code //

We have already seen the Worker code in the .js file. Below is the code for the HTML //
:page

<DOCTYPE html!> //
<html> //
<body> //

Count numbers: <output id="result"></output> //
<button onclick="startWorker()">Start Worker</button> //
<button onclick="stopWorker()">Stop Worker</button> //

<script> //
;let w //

} ()function startWorker //
} ("if (typeof(w) == "undefined //
;("w = new Worker("demo_workers.js //
{ //
} (w.onmessage = function(event //
;document.getElementById("result").innerHTML = event.data //
;{ //
{ //

} ()function stopWorker //
;()w.terminate //
;w = undefined //
{ //
<script/> //

<body/> //
<html/> //

JavaScript Fetch API //

A Fetch API Example //
:The example below fetches a file and displays the content //

(fetch(file //
(()then(x => x.text. //
;((then(y => myDisplay(y. //

Since Fetch is based on async and await, the example above might be easier to //
:understand like this

} (async function getText(file //
;(let x = await fetch(file //
;()let y = await x.text //
;(myDisplay(y //

```
{ //

:Or even better: Use understandable names instead of x and y //

} (async function getText(file //
;(let myObject = await fetch(file   //
;()let myText = await myObject.text   //
;(myDisplay(myText   //
{ //

Web Geolocation API //

Using the Geolocation API //
.The getCurrentPosition() method is used to return the user's position //

:The example below returns the latitude and longitude of the user's position //

<script> */}
;("const x = document.getElementById("demo
} ()function getLocation
} (if (navigator.geolocation
;(navigator.geolocation.getCurrentPosition(showPosition
} else {
;".x.innerHTML = "Geolocation is not supported by this browser
{
{

} (function showPosition(position
+ x.innerHTML = "Latitude: " + position.coords.latitude
;br>Longitude: " + position.coords.longitude>"
{
{/* <script/>

Handling Errors and Rejections //
The second parameter of the getCurrentPosition() method is used to handle errors. It //
:specifies a function to run if it fails to get the user's location

} (function showError(error //
} (switch(error.code   //
:case error.PERMISSION_DENIED     //
".x.innerHTML = "User denied the request for Geolocation       //
;break       //
:case error.POSITION_UNAVAILABLE     //
".x.innerHTML = "Location information is unavailable       //
;break       //
:case error.TIMEOUT     //
".x.innerHTML = "The request to get user location timed out       //
;break       //
```

:case error.UNKNOWN_ERROR   //
".x.innerHTML = "An unknown error occurred   //
;break   //
{   //
{ //

Displaying the Result in a Map //
.To display the result in a map, you need access to a map service, like Google Maps //

In the example below, the returned latitude and longitude is used to show the location in a //
:(Google Map (using a static image

} (function showPosition(position //
;let latlon = position.coords.latitude + "," + position.coords.longitude   //

=let img_url = "https://maps.googleapis.com/maps/api/staticmap?center   //
;"latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR_KEY+"   //

;"<'"+document.getElementById("mapholder").innerHTML = "<img src='"+img_url   //
{ //

Location-specific Information //
.This page has demonstrated how to show a user's position on a map //

:Geolocation is also very useful for location-specific information, like //

Up-to-date local information //
Showing Points-of-interest near the user //
(Turn-by-turn navigation (GPS //
The getCurrentPosition() Method - Return Data //
The getCurrentPosition() method returns an object on success. The latitude, longitude and //
:accuracy properties are always returned. The other properties are returned if available

Returns   Property //
(The latitude as a decimal number (always returned   coords.latitude //
(The longitude as a decimal number (always returned   coords.longitude //
(The accuracy of position (always returned   coords.accuracy //
(The altitude in meters above the mean sea level (returned if available   coords.altitude //
(The altitude accuracy of position (returned if available   coords.altitudeAccuracy //
(The heading as degrees clockwise from North (returned if available   coords.heading //
(The speed in meters per second (returned if available   coords.speed //
(The date/time of the response (returned if available   timestamp //
Geolocation Object - Other interesting Methods //
:The Geolocation object also has other interesting methods //

watchPosition() - Returns the current position of the user and continues to return updated //
.(position as the user moves (like the GPS in a car
.clearWatch() - Stops the watchPosition() method //

The example below shows the watchPosition() method. You need an accurate GPS device //
:(to test this (like smartphone

{/* <script> */}
;("const x = document.getElementById("demo //
} ()function getLocation //
} (if (navigator.geolocation //
;(navigator.geolocation.watchPosition(showPosition //
} else { //
;".x.innerHTML = "Geolocation is not supported by this browser //
{ //
{ //
} (function showPosition(position //
+ x.innerHTML = "Latitude: " + position.coords.latitude //
;br>Longitude: " + position.coords.longitude>" //
{ //
<script/> //

AJAX Introduction //

:AJAX is a developer's dream, because you can //

Read data from a web server - after the page has loaded //
Update a web page without reloading the page //
Send data to a web server - in the background //

<DOCTYPE html!> //
<html> //
<body> //

<"div id="demo> //
<h2>The XMLHttpRequest Object</h2> //
<button type="button" onclick="loadDoc()">Change Content</button> //
 //

<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
= document.getElementById("demo").innerHTML //
;this.responseText //
{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //
{ //
<script/> //

<body/> //

<html/> //

AJAX Example Explained //

HTML Page //
<DOCTYPE html!> //
<html> //
<body> //

<"div id="demo> //
<h2>Let AJAX change this text</h2> //
<button type="button" onclick="loadDoc()">Change Content</button> //
 //

<body/> //
<html/> //
.<The HTML page contains a section and a <button //

.The section is used to display information from a server //

.(The <button> calls a function (if it is clicked //

:The function requests data from a web server and displays it //

()Function loadDoc //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;document.getElementById("demo").innerHTML = this.responseText //
{ //
;(xhttp.open("GET", "ajax_info.txt", true //
;()xhttp.send //
{ //

?What is AJAX //
.AJAX = Asynchronous JavaScript And XML //

.AJAX is not a programming language //

:AJAX just uses a combination of //

(A browser built-in XMLHttpRequest object (to request data from a web server //
(JavaScript and HTML DOM (to display or use the data //
AJAX is a misleading name. AJAX applications might use XML to transport data, but it is //
.equally common to transport data as plain text or JSON text

AJAX allows web pages to be updated asynchronously by exchanging data with a web // server behind the scenes. This means that it is possible to update parts of a web page, .without reloading the whole page

(An event occurs in a web page (the page is loaded, a button is clicked .1 //
An XMLHttpRequest object is created by JavaScript .2 //
The XMLHttpRequest object sends a request to a web server .3 //
The server processes the request .4 //
The server sends a response back to the web page .5 //
The response is read by JavaScript .6 //
Proper action (like page update) is performed by JavaScript .7 //
(Modern Browsers (Fetch API //
.Modern Browsers can use Fetch API instead of the XMLHttpRequest Object //

.The Fetch API interface allows web browser to make HTTP requests to web servers //

.If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way //

AJAX - The XMLHttpRequest Object //

.The keystone of AJAX is the XMLHttpRequest object //

Create an XMLHttpRequest object //
Define a callback function //
Open the XMLHttpRequest object //
Send a Request to a server //
The XMLHttpRequest Object //
.All modern browsers support the XMLHttpRequest object //

The XMLHttpRequest object can be used to exchange data with a web server behind the // scenes. This means that it is possible to update parts of a web page, without reloading the .whole page

Create an XMLHttpRequest Object //
All modern browsers (Chrome, Firefox, IE, Edge, Safari, Opera) have a built-in // .XMLHttpRequest object

:Syntax for creating an XMLHttpRequest object //

;()variable = new XMLHttpRequest //
Define a Callback Function //
.A callback function is a function passed as a parameter to another function //

In this case, the callback function should contain the code to execute when the response is // .ready

} ()xhttp.onload = function //
What to do when the response is ready // //

{ //
Send a Request //
To send a request to a server, you can use the open() and send() methods of the //
:XMLHttpRequest object

;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //

<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //

<"div id="demo> //
Let AJAX change this text. //
<button type="button" onclick="loadDoc()">Change Content</button> //
 //

<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;document.getElementById("demo").innerHTML = this.responseText //
{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //
{ //
<script/> //

<body/> //
<html/> //

Access Across Domains //
.For security reasons, modern browsers do not allow access across domains //

This means that both the web page and the XML file it tries to load, must be located on the //
.same server

.The examples on W3Schools all open XML files located on the W3Schools domain //

If you want to use the example above on one of your own web pages, the XML files you //
.load must be located on your own server

XMLHttpRequest Object Methods //
Description Method //
Creates a new XMLHttpRequest object new XMLHttpRequest() //
Cancels the current request abort() //

Returns header information getAllResponseHeaders() //
Returns specific header information getResponseHeader() //
Specifies the request open(method, url, async, user, psw) //

method: the request type GET or POST //
url: the file location //
(async: true (asynchronous) or false (synchronous //
user: optional user name //
psw: optional password //
Sends the request to the server send() //
Used for GET requests //
.Sends the request to the server send(string) //
Used for POST requests //
Adds a label/value pair to the header to be sent setRequestHeader() //
XMLHttpRequest Object Properties //
Description Property //
(Defines a function to be called when the request is recieved (loaded onload //
Defines a function to be called when the readyState property onreadystatechange //
changes
.Holds the status of the XMLHttpRequest readyState //
request not initialized :0 //
server connection established :1 //
request received :2 //
processing request :3 //
request finished and response is ready :4 //
Returns the response data as a string responseText //
Returns the response data as XML data responseXML //
Returns the status-number of a request status //
"OK" :200 //
"Forbidden" :403 //
"Not Found" :404 //

For a complete list go to the Http Messages Reference //
("Returns the status-text (e.g. "OK" or "Not Found statusText //
The onload Property //
With the XMLHttpRequest object you can define a callback function to be executed when //
.the request receives an answer

:The function is defined in the onload property of the XMLHttpRequest object //

<DOCTYPE html!> //
<html> //
<body> //

<"div id="demo> //
<h2>The XMLHttpRequest Object</h2> //
<button type="button" onclick="loadDoc()">Change Content</button> //
 //

```
<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest   //
} ()xhttp.onload = function   //
= document.getElementById("demo").innerHTML     //
;this.responseText     //
{   //
;("xhttp.open("GET", "ajax_info.txt   //
;()xhttp.send   //
{ //
<script/> //

<body/> //
<html/> //
```

Multiple Callback Functions //
If you have more than one AJAX task in a website, you should create one function for //
.executing the XMLHttpRequest object, and one callback function for each AJAX task

The function call should contain the URL and what function to call when the response is //
.ready

```
;(loadDoc("url-1", myFunction1 //

;(loadDoc("url-2", myFunction2 //

} (function loadDoc(url, cFunction //
;()const xhttp = new XMLHttpRequest   //
{;(xhttp.onload = function() {cFunction(this   //
;(xhttp.open("GET", url   //
;()xhttp.send   //
{ //

} (function myFunction1(xhttp //
action goes here //   //
{ //
} (function myFunction2(xhttp //
action goes here //   //
{ //
```

The onreadystatechange Property //
.The readyState property holds the status of the XMLHttpRequest //

The onreadystatechange property defines a callback function to be executed when the //
.readyState changes

The status property and the statusText properties hold the status of the XMLHttpRequest //
.object

Description Property //
Defines a function to be called when the readyState property onreadystatechange //
changes
.Holds the status of the XMLHttpRequest readyState //
request not initialized :0 //
server connection established :1 //
request received :2 //
processing request :3 //
request finished and response is ready :4 //
"200: "OK status //
"Forbidden" :403 //
"Page not found" :404 //
For a complete list go to the Http Messages Reference //
("Returns the status-text (e.g. "OK" or "Not Found statusText //
.The onreadystatechange function is called every time the readyState changes //

:When readyState is 4 and status is 200, the response is ready //

} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onreadystatechange = function //
} (if (this.readyState == 4 && this.status == 200 //
= document.getElementById("demo").innerHTML //
;this.responseText //
{ //
;{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //
{ //

The onreadystatechange event is triggered four times (1-4), one time for each change in //
.the readyState

AJAX - XMLHttpRequest //

.The XMLHttpRequest object is used to request data from a server //

Send a Request To a Server //
To send a request to a server, we use the open() and send() methods of the //
:XMLHttpRequest object

;(xhttp.open("GET", "ajax_info.txt", true //
;()xhttp.send //
Description Method //
Specifies the type of request open(method, url, async) //

method: the type of request: GET or POST //
url: the server (file) location //
(async: true (asynchronous) or false (synchronous //
(Sends the request to the server (used for GET send() //
(Sends the request to the server (used for POST send(string) //
The url - A File On a Server //
:The url parameter of the open() method, is an address to a file on a server //

;(xhttp.open("GET", "ajax_test.asp", true //
The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php //
.((which can perform actions on the server before sending the response back

?Asynchronous - True or False //
.Server requests should be sent asynchronously //

:The async parameter of the open() method should be set to true //

;(xhttp.open("GET", "ajax_test.asp", true //
By sending asynchronously, the JavaScript does not have to wait for the server response, //
:but can instead

execute other scripts while waiting for server response //
deal with the response after the response is ready //
.The default value for the async parameter is async = true //

.You can safely remove the third parameter from your code //

Synchronous XMLHttpRequest (async = false) is not recommended because the //
JavaScript will stop executing until the server response is ready. If the server is busy or slow,
.the application will hang or stop

?GET or POST //
.GET is simpler and faster than POST, and can be used in most cases //

:However, always use POST requests when //

.(A cached file is not an option (update a file or database on the server //
.(Sending a large amount of data to the server (POST has no size limitations //
Sending user input (which can contain unknown characters), POST is more robust and //
.secure than GET
GET Requests //
:A simple GET request //

;("xhttp.open("GET", "demo_get.asp //
;()xhttp.send //

In the example above, you may get a cached result. To avoid this, add a unique ID to the //
:URL

;(()xhttp.open("GET", "demo_get.asp?t=" + Math.random //
;()xhttp.send //

:If you want to send information with the GET method, add the information to the URL //

;("xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford //
;()xhttp.send //

How the server uses the input and how the server responds to a request, is explained in a //
.later chapter

POST Requests //
:A simple POST request //

;("xhttp.open("POST", "demo_post.asp //
;()xhttp.send //

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify //
:the data you want to send in the send() method

<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //
<button type="button" onclick="loadDoc()">Request data</button> //

 //

<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;document.getElementById("demo").innerHTML = this.responseText //
{ //
;("xhttp.open("POST", "demo_post2.asp //
;("xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded //
;("xhttp.send("fname=Henry&lname=Ford //
{ //
<script/> //

<body/> //
<html/> //

Description Method //

Adds HTTP headers to the request setRequestHeader(header, value) //

header: specifies the header name //
value: specifies the header value //
Synchronous Request //
To execute a synchronous request, change the third parameter in the open() method to //
:false

;(xhttp.open("GET", "ajax_info.txt", false //
Sometimes async = false are used for quick testing. You will also find synchronous //
.requests in older JavaScript code

Since the code will wait for server completion, there is no need for an onreadystatechange //
:function

<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //

Let AJAX change this text. //

<button type="button" onclick="loadDoc()">Change Content</button> //

<script> //
} ()function loadDoc //
;()var xhttp = new XMLHttpRequest   //
;(xhttp.open("GET", "ajax_info.txt", false   //
;()xhttp.send   //
;document.getElementById("demo").innerHTML = xhttp.responseText   //
{ //
<script/> //

<body/> //
<html/> //

Synchronous XMLHttpRequest (async = false) is not recommended because the //
JavaScript will stop executing until the server response is ready. If the server is busy or slow,
.the application will hang or stop

Modern developer tools are encouraged to warn about using synchronous requests and //
.may throw an InvalidAccessError exception when it occurs

AJAX - Server Response //

Server Response Properties //
Description   Property //

get the response data as a string responseText //
get the response data as XML data responseXML //
The responseText Property //
The responseText property returns the server response as a JavaScript string, and you //
:can use it accordingly

```
<DOCTYPE html!> //
<html> //
<body> //

<"div id="demo> //
<h2>The XMLHttpRequest Object</h2> //
<button type="button" onclick="loadDoc()">Change Content</button> //
<div/> //

<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
= document.getElementById("demo").innerHTML //
;this.responseText //
{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //
{ //
<script/> //

<body/> //
<html/> //
```

The responseXML Property //
.The XMLHttpRequest object has an in-built XML parser //

.The responseXML property returns the server response as an XML DOM object //

:Using this property you can parse the response as an XML DOM object //

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //
<p id="demo"></p> //

<script> //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;const xmlDoc = this.responseXML //
```

```
;("const x = xmlDoc.getElementsByTagName("ARTIST   //
;"" = let txt   //
} (++for (let i = 0; i < x.length; i   //
;"<txt = txt + x[i].childNodes[0].nodeValue + "<br     //
{   //
;document.getElementById("demo").innerHTML = txt   //
{ //
;("xhttp.open("GET", "cd_catalog.xml //
;()xhttp.send //
<script/> //

<body/> //
<html/> //

Server Response Methods //
Description         Method //
Returns specific header information from the server resource         getResponseHeader() //
Returns all the header information from the server resource   getAllResponseHeaders() //
The getAllResponseHeaders() Method //
The getAllResponseHeaders() method returns all header information from the server //
.response

;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
= document.getElementById("demo").innerHTML     //
;()this.getAllResponseHeaders     //
{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //

The getResponseHeader() Method //
The getResponseHeader() method returns specific header information from the server //
.response

<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //

p>The getResponseHeader() function is used to return specific header information from a> //
<resource, like length, server-type, content-type, last-modified, etc:</p

<p>Last modified: <span id="demo"></span></p> //

<script> //
;()const xhttp=new XMLHttpRequest //
} ()xhttp.onload = function //
```

```
= document.getElementById("demo").innerHTML   //
;("this.getResponseHeader("Last-Modified   //
{ //
;("xhttp.open("GET", "ajax_info.txt //
;()xhttp.send //
<script/> //

<body/> //
<html/> //

AJAX XML Example //

.AJAX can be used for interactive communication with an XML file //

AJAX XML Example //
The following example will demonstrate how a web page can fetch information from an //
:XML file with AJAX

<DOCTYPE html!> //
<html> //
<style> //
} table,th,td //
;border : 1px solid black   //
;border-collapse: collapse   //
{ //
} th,td //
;padding: 5px   //
{ //
<style/> //
<body> //

<h2>The XMLHttpRequest Object</h2> //

<button type="button" onclick="loadDoc()">Get my CD collection</button> //
<br><br> //
<table id="demo"></table> //

<script> //
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest   //
} ()xhttp.onload = function   //
;(myFunction(this     //
{   //
;("xhttp.open("GET", "cd_catalog.xml   //
;()xhttp.send   //
{ //
} (function myFunction(xml //
;const xmlDoc = xml.responseXML   //
```

```
;("const x = xmlDoc.getElementsByTagName("CD   //
;"<let table="<tr><th>Artist</th><th>Title</th></tr   //
} (++for (let i = 0; i <x.length; i   //
+ "<table += "<tr><td     //
+ x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue     //
+ "<td><td/>"     //
+ x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue     //
;"<td></tr/>"     //
{   //
;document.getElementById("demo").innerHTML = table   //
{ //
<script/> //

<body/> //
<html/> //
```

Example Explained //
.When a user clicks on the "Get CD info" button above, the loadDoc() function is executed //

The loadDoc() function creates an XMLHttpRequest object, adds the function to be //
.executed when the server response is ready, and sends the request off to the server

When the server response is ready, an HTML table is built, nodes (elements) are extracted //
from the XML file, and it finally updates the element "demo" with the HTML table filled with
:XML data

```
} ()function loadDoc //
;()const xhttp = new XMLHttpRequest   //
{;(xhttp.onload = function() {myFunction(this   //
;("xhttp.open("GET", "cd_catalog.xml   //
;()xhttp.send   //
{ //
} (function myFunction(xml //
;const xmlDoc = xml.responseXML   //
;("const x = xmlDoc.getElementsByTagName("CD   //
;"<let table="<tr><th>Artist</th><th>Title</th></tr   //
} (++for (let i = 0; i <x.length; i   //
+ "<table += "<tr><td     //
+ x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue     //
+ "<td><td/>"     //
+ x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue     //
;"<td></tr/>"     //
{   //
;document.getElementById("demo").innerHTML = table   //
{ //
```

AJAX PHP Example //

Example Explained //
In the example above, when a user types a character in the input field, a function called //
.showHint() is executed

.The function is triggered by the onkeyup event //

:Here is the code //

<DOCTYPE html!> //
<html> //
<body> //

<h2>The XMLHttpRequest Object</h2> //
<h3>Start typing a name in the input field below:</h3> //

Suggestions: //
First name: <input type="text" id="txt1" onkeyup="showHint(this.value)"> //

<script> //
} (function showHint(str //
} (if (str.length == 0 //
;"" = document.getElementById("txtHint").innerHTML //
;return //
{ //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
= document.getElementById("txtHint").innerHTML //
;this.responseText //
{ //
;(xhttp.open("GET", "gethint.php?q="+str //
;()xhttp.send //
{ //
<script/> //

<body/> //
<html/> //

:Code explanation //

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint //
.placeholder and exit the function

:However, if the input field is not empty, do the following //

Create an XMLHttpRequest object //
Create the function to be executed when the server response is ready //
Send the request off to a PHP file (gethint.php) on the server //
Notice that q parameter is added gethint.php?q="+str //

The str variable holds the content of the input field //

"The PHP File - "gethint.php //
The PHP file checks an array of names, and returns the corresponding name(s) to the //
:browser

php?> //
Array with names //
;"a[] = "Anna$ //
;"a[] = "Brittany$ //
;"a[] = "Cinderella$ //
;"a[] = "Diana$ //
;"a[] = "Eva$ //
;"a[] = "Fiona$ //
;"a[] = "Gunda$ //
;"a[] = "Hege$ //
;"a[] = "Inga$ //
;"a[] = "Johanna$ //
;"a[] = "Kitty$ //
;"a[] = "Linda$ //
;"a[] = "Nina$ //
;"a[] = "Ophelia$ //
;"a[] = "Petunia$ //
;"a[] = "Amanda$ //
;"a[] = "Raquel$ //
;"a[] = "Cindy$ //
;"a[] = "Doris$ //
;"a[] = "Eve$ //
;"a[] = "Evita$ //
;"a[] = "Sunniva$ //
;"a[] = "Tove$ //
;"a[] = "Unni$ //
;"a[] = "Violet$ //
;"a[] = "Liza$ //
;"a[] = "Elizabeth$ //
;"a[] = "Ellen$ //
;"a[] = "Wenche$ //
;"a[] = "Vicky$ //

get the q parameter from URL // //
;["q = $_REQUEST["q$ //

;"" = hint$ //

"" lookup all hints from array if $q is different from // //
} ("" ==! if ($q //
;(q = strtolower(q //
;(len=strlen(q //

```
} (foreach($a as $name   //
} (((if (stristr($q, substr($name, 0, $len     //
} ("" === if ($hint       //
;hint = $name$         //
} else {       //
;"hint .= ", $name$         //
{       //
{     //
{   //
{ //

Output "no suggestion" if no hint was found or output correct values // //
;echo $hint === "" ? "no suggestion" : $hint //
<? //

AJAX Database Example //

.AJAX can be used for interactive communication with a database //

AJAX Database Example //
The following example will demonstrate how a web page can fetch information from a //
:database with AJAX

<DOCTYPE html!> //
<html> //
<style> //
} th,td //
;padding: 5px   //
{ //
<style/> //
<body> //

<h2>The XMLHttpRequest Object</h2> //

<""=form action> //
<"(select name="customers" onchange="showCustomer(this.value>   //
<option value="">Select a customer:</option>     //
<option value="ALFKI">Alfreds Futterkiste</option>     //
<option value="NORTS ">North/South</option>     //
<option value="WOLZA">Wolski Zajazd</option>     //
<select/>   //
<form/> //
<br> //
<div id="txtHint">Customer info will be listed here...</div> //

<script> //
} (function showCustomer(str //
} ("" == if (str   //
```

;"" = document.getElementById("txtHint").innerHTML //
;return //
{ //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;document.getElementById("txtHint").innerHTML = this.responseText //
{ //
;(xhttp.open("GET", "getcustomer.php?q="+str //
;()xhttp.send //
{ //
<script/> //
<body/> //
<html/> //

Example Explained - The showCustomer() Function //
When a user selects a customer in the dropdown list above, a function called //
:showCustomer() is executed. The function is triggered by the onchange event

showCustomer //
} (function showCustomer(str //
} ("" == if (str //
;"" = document.getElementById("txtHint").innerHTML //
;return //
{ //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;document.getElementById("txtHint").innerHTML = this.responseText //
{ //
;(xhttp.open("GET", "getcustomer.php?q="+str //
;()xhttp.send //
{ //
:The showCustomer() function does the following //

Check if a customer is selected //
Create an XMLHttpRequest object //
Create the function to be executed when the server response is ready //
Send the request off to a file on the server //
(Notice that a parameter (q) is added to the URL (with the content of the dropdown list //

The AJAX Server Page //
The page on the server called by the JavaScript above is a PHP file called //
."getcustomer.php

The source code in "getcustomer.php" runs a query against a database, and returns the //
:result in an HTML table

php?> //
;("mysqli = new mysqli("servername", "username", "password", "dbname$ //

```
} (if($mysqli->connect_error //
;('exit('Could not connect   //
{ //

sql = "SELECT customerid, companyname, contactname, address, city, postalcode,$ //
country
;"? = FROM customers WHERE customerid //

;(stmt = $mysqli->prepare($sql$ //
;(['stmt->bind_param("s", $_GET['q$ //
;()stmt->execute$ //
;()stmt->store_result$ //
;(stmt->bind_result($cid, $cname, $name, $adr, $city, $pcode, $country$ //
;()stmt->fetch$ //
;()stmt->close$ //

;"<echo "<table //
;"<echo "<tr //
;"<echo "<th>CustomerID</th //
;"<echo "<td>" . $cid . "</td //
;"<echo "<th>CompanyName</th //
;"<echo "<td>" . $cname . "</td //
;"<echo "<th>ContactName</th //
;"<echo "<td>" . $name . "</td //
;"<echo "<th>Address</th //
;"<echo "<td>" . $adr . "</td //
;"<echo "<th>City</th //
;"<echo "<td>" . $city . "</td //
;"<echo "<th>PostalCode</th //
;"<echo "<td>" . $pcode . "</td //
;"<echo "<th>Country</th //
;"<echo "<td>" . $country . "</td //
;"<echo "</tr //
;"<echo "</table //
<? //
```

XML Applications //

This chapter demonstrates some HTML applications using XML, HTTP, DOM, and //
.JavaScript

The XML Document Used //
."In this chapter we will use the XML file called "cd_catalog.xml //

Display XML Data in an HTML Table //
This example loops through each <CD> element, and displays the values of the <ARTIST> //
:and the <TITLE> elements in an HTML table

```
<DOCTYPE html!> //
<html> //
<style> //
} table,th,td //
;border : 1px solid black   //
;border-collapse: collapse   //
{ //
} th,td //
;padding: 5px   //
{ //
<style/> //
<body> //

<button type="button" onclick="loadXMLDoc()">Get my CD collection</button> //
<br><br> //
<table id="demo"></table> //

<script> //
} ()function loadXMLDoc //
;()const xhttp = new XMLHttpRequest   //
} ()xhttp.onload = function   //
;const xmlDoc = xhttp.responseXML     //
;("const cd = xmlDoc.getElementsByTagName("CD     //
(myFunction(cd     //
{   //
;("xhttp.open("GET", "cd_catalog.xml   //
;()xhttp.send   //
{ //

} (function myFunction(cd //
;"<let table="<tr><th>Artist</th><th>Title</th></tr   //
} (++for (let i = 0; i < cd.length; i   //
+ "<table += "<tr><td     //
+ cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue     //
+ "<td><td/>"     //
+ cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue     //
;"<td></tr/>"     //
{   //
;document.getElementById("demo").innerHTML = table   //
{ //
<script/> //

<body/> //
<html/> //

Display the First CD in an HTML div Element //
This example uses a function to display the first CD element in an HTML element with //
:"id="showCD
```

```
<DOCTYPE html!> //
<html> //
<body> //

<div id='showCD'></div> //

<script> //
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;const xmlDoc = xhttp.responseXML   //
;("const cd = xmlDoc.getElementsByTagName("CD   //
;(myFunction(cd, 0   //
{ //
;("xhttp.open("GET", "cd_catalog.xml //
;()xhttp.send //

} (function myFunction(cd, i //
= document.getElementById("showCD").innerHTML   //
+ " :Artist"   //
+ cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue   //
+ " :br>Title>"   //
+ cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue   //
+ " :br>Year>"   //
;cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue   //
{ //
<script/> //

<body/> //
<html/> //
```

Navigate Between the CDs //
To navigate between the CDs in the example above, create a next() and previous() //
:function

```
<DOCTYPE html!> //
<html> //
<body> //

<div id='showCD'></div><br> //
<">>"=input type="button" onclick="previous()" value> //
<"<<"=input type="button" onclick="next()" value> //

<script> //
;let i = 0 //
;let len //
;let cd //
```

```
;()const xhttp = new XMLHttpRequest //
} ()xhttp.onload = function //
;const xmlDoc = xhttp.responseXML   //
;("cd = xmlDoc.getElementsByTagName("CD   //
;len = cd.length   //
;(displayCD(i   //
{ //
;("xhttp.open("GET", "cd_catalog.xml //
;()xhttp.send //

} (function displayCD(i //
= document.getElementById("showCD").innerHTML   //
+ " :Artist"   //
+ cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue   //
+ " :br>Title>"   //
+ cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue   //
+ " :br>Year>"   //
;cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue   //
{ //

} ()function next //
} (if (i < len-1   //
;++i     //
;(displayCD(i     //
{   //
{ //

} ()function previous //
} (if (i > 0   //
;--i     //
;(displayCD(i     //
{   //
{ //
<script/> //

<body/> //
<html/> //

Show Album Information When Clicking On a CD //
The last example shows how you can show album information when the user clicks on a //
:CD

<DOCTYPE html!> //
<html> //

<head> //
<style> //
} table,th,td //
```

```
;border : 1px solid black   //
;border-collapse: collapse   //
{ //
} th,td //
;padding: 5px   //
{ //
<style/> //
<head/> //

<body> //

<p>Click on a CD to display album information.</p> //
<p id='showCD'></p> //
<table id="demo"></table> //

<script> //
;()const xhttp = new XMLHttpRequest //
;let cd //
} ()xhttp.onload = function //
;const xmlDoc = xhttp.responseXML  //
;("cd = xmlDoc.getElementsByTagName("CD  //
;()loadCD  //
{ //
;("xhttp.open("GET", "cd_catalog.xml //
;()xhttp.send //

} ()function loadCD //
;"<let table="<tr><th>Artist</th><th>Title</th></tr   //
} (++for (let i = 0; i < cd.length; i   //
;"<table += "<tr onclick='displayCD(" + i + ")'><td     //
;table += cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue     //
;"<table += "</td><td     //
;table += cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue     //
;"<table += "</td></tr     //
{   //
;document.getElementById("demo").innerHTML = table   //
{ //

} (function displayCD(i //
= document.getElementById("showCD").innerHTML   //
+ " :Artist"   //
+ cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue   //
+ " :br>Title>"   //
+ cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue   //
+ " :br>Year>"   //
;cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue   //
{ //
<script/> //
```

<body/> //
<html/> //

JSON - Introduction //

JSON stands for JavaScript Object Notation //

JSON is a text format for storing and transporting data //

JSON is "self-describing" and easy to understand //

JSON Example //
:This example is a JSON string //

'{name":"John", "age":30, "car":null"}' //
:It defines an object with 3 properties //

name //
age //
car //
.Each property has a value //

If you parse the JSON string with a JavaScript program, you can access the data as an //
:object

;let personName = obj.name //
;let personAge = obj.age //
?What is JSON //
JSON stands for JavaScript Object Notation //
JSON is a lightweight data-interchange format //
JSON is plain text written in JavaScript object notation //
JSON is used to send data between computers //
* JSON is language independent //
* //
The JSON syntax is derived from JavaScript object notation, but the JSON format is text //
.only

.Code for reading and generating JSON exists in many programming languages //

?Why Use JSON //
The JSON format is syntactically similar to the code for creating JavaScript objects. //
.Because of this, a JavaScript program can easily convert JSON data into JavaScript objects

Since the format is text only, JSON data can easily be sent between computers, and used //
.by any programming language

:JavaScript has a built in function for converting JSON strings into JavaScript objects //

()JSON.parse //

:JavaScript also has a built in function for converting an object into a JSON string //

()JSON.stringify //

.You can receive pure text from a server and use it as a JavaScript object //

.You can send a JavaScript object to a server in pure text format //

You can work with data as JavaScript objects, with no complicated parsing and //
.translations

Storing Data //
When storing data, the data has to be a certain format, and regardless of where you //
.choose to store it, text is always one of the legal formats

.JSON makes it possible to store JavaScript objects as text //

JSON Syntax //

.The JSON syntax is a subset of the JavaScript syntax //

JSON Syntax Rules //
:JSON syntax is derived from JavaScript object notation syntax //

Data is in name/value pairs //
Data is separated by commas //
Curly braces hold objects //
Square brackets hold arrays //
JSON Data - A Name and a Value //
.(JSON data is written as name/value pairs (aka key/value pairs //

A name/value pair consists of a field name (in double quotes), followed by a colon, //
:followed by a value

Example //
"name":"John" //
.JSON names require double quotes //

JSON - Evaluates to JavaScript Objects //
.The JSON format is almost identical to JavaScript objects //

:In JSON, keys must be strings, written with double quotes //

JSON //
{"name":"John"} //

:In JavaScript, keys can be strings, numbers, or identifier names //

JavaScript //
{"name:"John} //

JSON Values //
:In JSON, values must be one of the following data types //

a string //
a number //
an object //
an array //
a boolean //
null //
In JavaScript values can be all of the above, plus any other valid JavaScript expression, //
:including

a function //
a date //
undefined //
:In JSON, string values must be written with double quotes //

JSON //
{"name":"John"} //
:In JavaScript, you can write string values with double or single quotes //

JavaScript //
{'name:'John} //
JavaScript Objects //
Because JSON syntax is derived from JavaScript object notation, very little extra software //
.is needed to work with JSON within JavaScript

:With JavaScript you can create an object and assign data to it, like this //

Example //
;{"person = {name:"John", age:31, city:"New York //
:You can access a JavaScript object like this //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Access a JavaScript object</h2> //
 //

<script> //
;{"const myObj = {name:"John", age:30, city:"New York //
;document.getElementById("demo").innerHTML = myObj.name //

<script/> //

<body/> //
<html/> //

:It can also be accessed like this //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Access a JavaScript object</h2> //
 //

<script> //
;{"const myObj = {name:"John", age:30, city:"New York //
;["document.getElementById("demo").innerHTML = myObj["name //
<script/> //

<body/> //
<html/> //

:Data can be modified like this //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Modify a JavaScript Object</h2> //
 //

<script> //
;{ "const myObj = {name:"John", age:30, city:"New York //
;"myObj.name = "Gilbert //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

:It can also be modified like this //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Modify a JavaScript Object</h2> //
 //

<script> //
;{"const myObj = {name:"John", age:30, city:"New York //
;"myObj["name"] = "Gilbert //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

.You will learn how to convert JavaScript objects into JSON later in this tutorial //

JavaScript Arrays as JSON //
The same way JavaScript objects can be written as JSON, JavaScript arrays can also be //
.written as JSON

.You will learn more about objects and arrays later in this tutorial //

JSON Files //
"The file type for JSON files is ".json //
"The MIME type for JSON text is "application/json //

JSON vs XML //

.Both JSON and XML can be used to receive data from a web server //

The following JSON and XML examples both define an employees object, with an array of //
:3 employees

JSON Example //
]:"employees"} //
,{ "firstName":"John", "lastName":"Doe" } //
,{ "firstName":"Anna", "lastName":"Smith" } //
{ "firstName":"Peter", "lastName":"Jones" } //
{[//
XML Example //
<employees> //
<employee> //
<firstName>John</firstName> <lastName>Doe</lastName> //
<employee/> //
<employee> //
<firstName>Anna</firstName> <lastName>Smith</lastName> //
<employee/> //
<employee> //
<firstName>Peter</firstName> <lastName>Jones</lastName> //
<employee/> //
<employees/> //
JSON is Like XML Because //

(Both JSON and XML are "self describing" (human readable //
(Both JSON and XML are hierarchical (values within values //
Both JSON and XML can be parsed and used by lots of programming languages //
Both JSON and XML can be fetched with an XMLHttpRequest //
JSON is Unlike XML Because //
JSON doesn't use end tag //
JSON is shorter //
JSON is quicker to read and write //
JSON can use arrays //
:The biggest difference is //

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript //
.function

Why JSON is Better Than XML //
.XML is much more difficult to parse than JSON //
.JSON is parsed into a ready-to-use JavaScript object //

:For AJAX applications, JSON is faster and easier than XML //

Using XML //

Fetch an XML document //
Use the XML DOM to loop through the document //
Extract values and store in variables //
Using JSON //

Fetch a JSON string //
JSON.Parse the JSON string //

JSON Data Types //

Valid Data Types //
:In JSON, values must be one of the following data types //

a string //
a number //
(an object (JSON object //
an array //
a boolean //
null //
:JSON values cannot be one of the following data types //

a function //
a date //
undefined //
JSON Strings //
.Strings in JSON must be written in double quotes //

Example //
{"name":"John"} //
JSON Numbers //
.Numbers in JSON must be an integer or a floating point //

Example //
{age":30"} //
JSON Objects //
.Values in JSON can be objects //

Example //
} //
{"employee":{"name":"John", "age":30, "city":"New York" //
{ //
.Objects as values in JSON must follow the JSON syntax //

JSON Arrays //
.Values in JSON can be arrays //

Example //
} //
["employees":["John", "Anna", "Peter" //
{ //
JSON Booleans //
.Values in JSON can be true/false //

Example //
{sale":true"} //
JSON null //
.Values in JSON can be null //

Example //
{middlename":null"} //

()JSON.parse //

.A common use of JSON is to exchange data to/from a web server //

.When receiving data from a web server, the data is always a string //

.Parse the data with JSON.parse(), and the data becomes a JavaScript object //

Example - Parsing JSON //
:Imagine we received this text from a web server //

'{"name":"John", "age":30, "city":"New York"}' //
:Use the JavaScript function JSON.parse() to convert text into a JavaScript object //

;('{"const obj = JSON.parse('{"name":"John", "age":30, "city":"New York //
.Make sure the text is in JSON format, or else you will get a syntax error //

:Use the JavaScript object in your page //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Creating an Object from a JSON String</h2> //

 //

<script> //
'{"const txt = '{"name":"John", "age":30, "city":"New York //
;(const obj = JSON.parse(txt //
;document.getElementById("demo").innerHTML = obj.name + ", " + obj.age //
<script/> //

<body/> //
<html/> //

Array as JSON //
When using the JSON.parse() on a JSON derived from an array, the method will return a //
.JavaScript array, instead of a JavaScript object

<DOCTYPE html!> //
<html> //
<body> //

<h2>Parsing a JSON Array.</h2> //
Data written as an JSON array will be parsed into a JavaScript array. //
 //

<script> //
;'["const text = '["Ford", "BMW", "Audi", "Fiat //
;(const myArr = JSON.parse(text //
;[document.getElementById("demo").innerHTML = myArr[0 //
<script/> //

<body/> //
<html/> //

Exceptions //
Parsing Dates //
.Date objects are not allowed in JSON //

.If you need to include a date, write it as a string //

:You can convert it back into a date object later //

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>Convert a string into a date object.</h2> //
<p id="demo"></p> //

<script> //
;'{"const text = '{"name":"John", "birth":"1986-12-14", "city":"New York //
;(const obj = JSON.parse(text //
;(obj.birth = new Date(obj.birth //
;document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth //
<script/> //

<body/> //
<html/> //
```

.Or, you can use the second parameter, of the JSON.parse() function, called reviver //

.The reviver parameter is a function that checks each property, before returning the value //

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>Convert a string into a date object.</h2> //

<p id="demo"></p> //

<script> //
;'{"const text = '{"name":"John", "birth":"1986-12-14", "city":"New York //
} (const obj = JSON.parse(text, function (key, value //
} ("if (key == "birth   //
;(return new Date(value     //
} else {   //
;return value     //
{   //
;({ //
;document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth //
<script/> //

<body/> //
<html/> //
```

Parsing Functions //
.Functions are not allowed in JSON //

.If you need to include a function, write it as a string //

:You can convert it back into a function later //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Convert a string into a function.</h2> //
 //

<script> //
;'{"const text = '{"name":"John", "age":"function() {return 30;}", "city":"New York //
;(const obj = JSON.parse(text //
;("(" + obj.age = eval("(" + obj.age //
;()document.getElementById("demo").innerHTML = obj.name + ", " + obj.age //
<script/> //

<body/> //
<html/> //

You should avoid using functions in JSON, the functions will lose their scope, and you //
.would have to use eval() to convert them back into functions

()JSON.stringify //

.A common use of JSON is to exchange data to/from a web server //

.When sending data to a web server, the data has to be a string //

.()Convert a JavaScript object into a string with JSON.stringify //

Stringify a JavaScript Object //
:Imagine we have this object in JavaScript //

;{"const obj = {name: "John", age: 30, city: "New York //
.Use the JavaScript function JSON.stringify() to convert it into a string //

;(const myJSON = JSON.stringify(obj //
.The result will be a string following the JSON notation //

:myJSON is now a string, and ready to be sent to a server //

<DOCTYPE html!> //
<html> //

<body> //

<h2>Create a JSON string from a JavaScript object.</h2> //
 //

<script> //
;{"const obj = {name: "John", age: 30, city: "New York //
;(const myJSON = JSON.stringify(obj //
;document.getElementById("demo").innerHTML = myJSON //
<script/> //

<body/> //
<html/> //

.You will learn how to send JSON to a server in the next chapters //

Stringify a JavaScript Array //
:It is also possible to stringify JavaScript arrays //

:Imagine we have this array in JavaScript //

;["const arr = ["John", "Peter", "Sally", "Jane //
.Use the JavaScript function JSON.stringify() to convert it into a string //

;(const myJSON = JSON.stringify(arr //
.The result will be a string following the JSON notation //

:myJSON is now a string, and ready to be sent to a server //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Create a JSON string from a JavaScript array.</h2> //
 //

<script> //
;["const arr = ["John", "Peter", "Sally", "Jane //
;(const myJSON = JSON.stringify(arr //
;document.getElementById("demo").innerHTML = myJSON //
<script/> //

<body/> //
<html/> //

.You will learn how to send a JSON string to a server in the next chapters //

Storing Data //

When storing data, the data has to be a certain format, and regardless of where you //
.choose to store it, text is always one of the legal formats

.JSON makes it possible to store JavaScript objects as text //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Store and retrieve data from local storage.</h2> //
 //

<script> //
:Storing data // //
;{ "const myObj = { name: "John", age: 31, city: "New York //
;(const myJSON = JSON.stringify(myObj //
;(localStorage.setItem("testJSON", myJSON //

:Retrieving data // //
;("let text = localStorage.getItem("testJSON //
;(let obj = JSON.parse(text //
;document.getElementById("demo").innerHTML = obj.name //
<script/> //

<body/> //
<html/> //

Exceptions //
Stringify Dates //
In JSON, date objects are not allowed. The JSON.stringify() function will convert any dates //
.into strings

<DOCTYPE html!> //
<html> //
<body> //

<h2>JSON.stringify() converts date objects into strings.</h2> //
 //

<script> //
;{"const obj = {name: "John", today: new Date(), city: "New York //
;(const myJSON = JSON.stringify(obj //
;document.getElementById("demo").innerHTML = myJSON //
<script/> //

<body/> //
<html/> //

This can be omitted if you convert your functions into strings before running the //
.JSON.stringify() function

<DOCTYPE html!> //
<html> //
<body> //

<h2>JSON.stringify() will remove any functions from an object.</h2> //
Convert functions into strings to keep them in the JSON object. //

 //

<script> //
;{"const obj = {name: "John", age: function () {return 30;}, city: "New York //
;()obj.age = obj.age.toString //
;(const myJSON = JSON.stringify(obj //
;document.getElementById("demo").innerHTML = myJSON //
<script/> //

<body/> //
<html/> //

If you send functions using JSON, the functions will lose their scope, and the receiver //
.would have to use eval() to convert them back into functions

JSON Object Literals //

:This is a JSON string //

'{name":"John", "age":30, "car":null"}' //
:Inside the JSON string there is a JSON object literal //

{name":"John", "age":30, "car":null"} //
.{} JSON object literals are surrounded by curly braces //

.JSON object literals contains key/value pairs //

.Keys and values are separated by a colon //

:Keys must be strings, and values must be a valid JSON data type //

string //
number //
object //
array //
boolean //
null //
.Each key/value pair is separated by a comma //

."It is a common mistake to call a JSON object literal "a JSON object //

.JSON cannot be an object. JSON is a string format //

The data is only JSON when it is in a string format. When it is converted to a JavaScript //
.variable, it becomes a JavaScript object

JavaScript Objects //
:You can create a JavaScript object from a JSON object literal //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Creating an Object from a JSON Literal</h2> //
 //

<script> //
;{const myObj = {"name":"John", "age":30, "car":null //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

:Normally, you create a JavaScript object by parsing a JSON string //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Creating an Object Parsing JSON</h2> //
 //

<script> //
;'{const myJSON = '{"name":"John", "age":30, "car":null //
;(const myObj = JSON.parse(myJSON //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

Accessing Object Values //
:You can access object values by using dot (.) notation //

<DOCTYPE html!> //

```
<html> //
<body> //

<h2>Access a JavaScript Object</h2> //
<p id="demo"></p> //

<script> //
;'{const myJSON = '{"name":"John", "age":30, "car":null //
;(const myObj = JSON.parse(myJSON //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

:You can also access object values by using bracket ([]) notation //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Access a JavaScript Object</h2> //
<p id="demo"></p> //

<script> //
;'{const myJSON = '{"name":"John", "age":30, "car":null //
;(const myObj = JSON.parse(myJSON //
;["document.getElementById("demo").innerHTML = myObj["name //
<script/> //

<body/> //
<html/> //

Looping an Object //
:You can loop through object properties with a for-in loop //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Looping Object Properties</h2> //
<p id="demo"></p> //

<script> //
;'{const myJSON = '{"name":"John", "age":30, "car":null //
;(const myObj = JSON.parse(myJSON //

;"" = let text //
```

} (for (const x in myObj //
;" ," + text += x   //
{ //
;document.getElementById("demo").innerHTML = text //
<script/> //

<body/> //
<html/> //

:In a for-in loop, use the bracket notation to access the property values //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Looping JavaScript Object Values</h2> //
 //

<script> //
;'{const myJSON = '{"name":"John", "age":30, "car":null //
;(const myObj = JSON.parse(myJSON //

;"" = let text //
} (for (const x in myObj //
;" ," + [text += myObj[x   //
{ //
;document.getElementById("demo").innerHTML = text //
<script/> //

<body/> //
<html/> //

JSON Array Literals //

:This is a JSON string //

'["Ford", "BMW", "Fiat"]' //
:Inside the JSON string there is a JSON array literal //

["Ford", "BMW", "Fiat"] //
.Arrays in JSON are almost the same as arrays in JavaScript //

.In JSON, array values must be of type string, number, object, array, boolean or null //

In JavaScript, array values can be all of the above, plus any other valid JavaScript //
.expression, including functions, dates, and undefined

JavaScript Arrays //

:You can create a JavaScript array from a literal //

<DOCTYPE html!> //
<html> //
<body> //
<h2>Creating an Array from a Literal</h2> //
 //

<script> //
;["const myArray = ["Ford", "BMW", "Fiat //
;document.getElementById("demo").innerHTML = myArray //
<script/> //

<body/> //
<html/> //

:You can create a JavaScript array by parsing a JSON string //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Creating an Array from JSON</h2> //
 //

<script> //
;'["const myJSON = '["Ford", "BMW", "Fiat //
;(const myArray = JSON.parse(myJSON //
;document.getElementById("demo").innerHTML = myArray //
<script/> //

<body/> //
<html/> //

Accessing Array Values //
:You access array values by index //

<DOCTYPE html!> //
<html> //
<body> //

<h1>Access an Array by Index</h1> //
 //

<script> //
;'["const myJSON = '["Ford", "BMW", "Fiat //
;(const myArray = JSON.parse(myJSON //
;[document.getElementById("demo").innerHTML = myArray[0 //

<script/> //

<body/> //
<html/> //

Arrays in Objects //
:Objects can contain arrays //

} //
,"name":"John" //
,age":30" //
["cars":["Ford", "BMW", "Fiat" //
{ //
:You access array values by index //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Access Array Values</h2> //
 //

<script> //
;'{["const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat //
;(const myObj = JSON.parse(myJSON //

;[document.getElementById("demo").innerHTML = myObj.cars[0 //
<script/> //

<body/> //
<html/> //

Looping Through an Array //
:You can access array values by using a for in loop //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Looping an Array</h2> //
 //

<script> //
;'{["const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat //
;(const myObj = JSON.parse(myJSON //

;"" = let text //
} (for (let i in myObj.cars //

;" ," + [text += myObj.cars[i   //
{ //

;document.getElementById("demo").innerHTML = text //
<script/> //

<body/> //
<html/> //

:Or you can use a for loop //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Looping an Array</h2> //
 //

<script> //
;'{["const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat //
;(const myObj = JSON.parse(myJSON //

;"" = let text //
} (++for (let i = 0; i < myObj.cars.length; i //
;" ," + [text += myObj.cars[i   //
{ //

;document.getElementById("demo").innerHTML = text //
<script/> //

<body/> //
<html/> //

JSON Server //

.A common use of JSON is to exchange data to/from a web server //

.When receiving data from a web server, the data is always a string //

.Parse the data with JSON.parse(), and the data becomes a JavaScript object //

Sending Data //
If you have data stored in a JavaScript object, you can convert the object into JSON, and //
:send it to a server

<DOCTYPE html!> //
<html> //
<body> //

<h2>Convert a JavaScript object into a JSON string, and send it to the server.</h2> //

<script> //
;{ "const myObj = { name: "John", age: 31, city: "New York //
;(const myJSON = JSON.stringify(myObj //
;window.location = "demo_json.php?x=" + myJSON //
<script/> //

<body/> //
<html/> //

Receiving Data //
:If you receive data in JSON format, you can easily convert it into a JavaScript object //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Convert a JSON string into a JavaScript object.</h2> //

 //

<script> //
;'{"const myJSON = '{"name":"John", "age":31, "city":"New York //
;(const myObj = JSON.parse(myJSON //
;document.getElementById("demo").innerHTML = myObj.name //
<script/> //

<body/> //
<html/> //

JSON From a Server //
You can request JSON from the server by using an AJAX request //

As long as the response from the server is written in JSON format, you can parse the //
.string into a JavaScript object

<DOCTYPE html!> //
<html> //
<body> //

<h2>Fetch a JSON file with XMLHttpRequest</h2> //
 //

<script> //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //

```
;(const myObj = JSON.parse(this.responseText   //
;document.getElementById("demo").innerHTML = myObj.name   //
{ //
;("xmlhttp.open("GET", "json_demo.txt //
;()xmlhttp.send //
<script/> //

<body/> //
<html/> //
```

Array as JSON //
When using the JSON.parse() on JSON derived from an array, the method will return a //
.JavaScript array, instead of a JavaScript object

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>Fetch a JSON file with XMLHttpRequest</h2> //
<p>Content written as an JSON array will be converted into a JavaScript array.</p> //
<p id="demo"></p> //

<script> //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(const myArr = JSON.parse(this.responseText   //
;[document.getElementById("demo").innerHTML = myArr[0   //
{ //
;(xmlhttp.open("GET", "json_demo_array.txt", true //
;()xmlhttp.send //
<script/> //

<body/> //
<html/> //
```

JSON PHP //

A common use of JSON is to read data from a web server, and display the data in a web //
.page

This chapter will teach you how to exchange JSON data between the client and a PHP //
.server

The PHP File //
.PHP has some built-in functions to handle JSON //

:()Objects in PHP can be converted into JSON by using the PHP function json_encode //

PHP file //

php?> //
;"myObj->name = "John$ //
;myObj->age = 30$ //
;"myObj->city = "New York$ //

;(myJSON = json_encode($myObj$ //

;echo $myJSON //
<? //

The Client JavaScript //
Here is a JavaScript on the client, using an AJAX call to request the PHP file from the //
:example above

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get JSON Data from a PHP Server</h2> //
 //

<script> //
;()const xmlhttp = new XMLHttpRequest //

} ()xmlhttp.onload = function //
;(const myObj = JSON.parse(this.responseText //
;document.getElementById("demo").innerHTML = myObj.name //
{ //
;("xmlhttp.open("GET", "demo_file.php //
;()xmlhttp.send //
<script/> //

<body/> //
<html/> //

PHP Array //
Arrays in PHP will also be converted into JSON when using the PHP function //
:()json_encode

PHP file //
php?> //
;("myArr = array("John", "Mary", "Peter", "Sally$ //

;(myJSON = json_encode($myArr$ //

;echo $myJSON //

<? //

The Client JavaScript //
Here is a JavaScript on the client, using an AJAX call to request the PHP file from the //
:array example above

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get JSON Data from a PHP Server</h2> //
Convert the data into a JavaScript array: //
 //

<script> //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(const myObj = JSON.parse(this.responseText //
;[document.getElementById("demo").innerHTML = myObj[2 //
{ //
;("xmlhttp.open("GET", "demo_file_array.php //
;()xmlhttp.send //
<script/> //

<body/> //
<html/> //

PHP Database //
.PHP is a server side programming language, and can be used to access a database //

Imagine you have a database on your server, and you want to send a request to it from the //
."client where you ask for the 10 first rows in a table called "customers

.On the client, make a JSON object that describes the numbers of rows you want to return //

Before you send the request to the server, convert the JSON object into a string and send //
:it as a parameter to the url of the PHP page

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get JSON Data from a PHP Server</h2> //
The JSON received from the PHP file: //

 //

<script> //

```
;({const dbParam = JSON.stringify({"limit":10 //

;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;document.getElementById("demo").innerHTML = this.responseText   //
{ //
;(xmlhttp.open("GET", "json_demo_db.php?x=" + dbParam //
;()xmlhttp.send //
<script/> //

<body/> //
<html/> //

:Example explained //
.Define an object containing a "limit" property and value //
.Convert the object into a JSON string //
.Send a request to the PHP file, with the JSON string as a parameter //
(Wait until the request returns with the result (as JSON //
.Display the result received from the PHP file //
:Take a look at the PHP file //

PHP file //
php?> //
;("header("Content-Type: application/json; charset=UTF-8 //
;(obj = json_decode($_GET["x"], false$ //

;("conn = new mysqli("myServer", "myUser", "myPassword", "Northwind$ //
;("? stmt = $conn->prepare("SELECT name FROM customers LIMIT$ //
;(stmt->bind_param("s", $obj->limit$ //
;()stmt->execute$ //
;()result = $stmt->get_result$ //
;(outp = $result->fetch_all(MYSQLI_ASSOC$ //

;(echo json_encode($outp //
<? //
:PHP File explained //
.()Convert the request into an object, using the PHP function json_decode //
.Access the database, and fill an array with the requested data //
Add the array to an object, and return the object as JSON using the json_encode() //
.function
Use the Data //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get JSON Data from a PHP Server</h2> //
<p id="demo"></p> //
```

```
<script> //
;{ const obj = { "limit":10 //
;(const dbParam = JSON.stringify(obj //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(myObj = JSON.parse(this.responseText   //
"" = let text   //
} (for (let x in myObj   //
;"<text += myObj[x].name + "<br     //
{   //
;document.getElementById("demo").innerHTML = text   //
;{ //
;(xmlhttp.open("GET", "json_demo_db.php?x=" + dbParam //
;()xmlhttp.send //
<script/> //

<p>Try changing the "limit" property from 10 to 5.</p> //

<body/> //
<html/> //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Use HTTP POST to Get JSON Data from a PHP Server</h2> //

<p id="demo"></p> //

<script> //
;({const dbParam = JSON.stringify({"limit":10 //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(myObj = JSON.parse(this.responseText   //
;"" = let text   //
} (for (let x in myObj   //
;"<text += myObj[x].name + "<br     //
{   //
;document.getElementById("demo").innerHTML = text   //
{ //
;("xmlhttp.open("POST", "json_demo_db_post.php //
;("xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded //
;(xmlhttp.send("x=" + dbParam //
<script/> //

<p>Try changing the "limit" property from 10 to 5.</p> //
```

<body/> //
<html/> //

.The only difference in the PHP file is the method for getting the transferred data //

PHP file //
:Use $_POST instead of $_GET //

php?> //
;("header("Content-Type: application/json; charset=UTF-8 //
;(obj = json_decode($_POST["x"], false$ //

;("conn = new mysqli("myServer", "myUser", "myPassword", "Northwind$ //
;("? stmt = $conn->prepare("SELECT name FROM customers LIMIT$ //
;(stmt->bind_param("s", $obj->limit$ //
;()stmt->execute$ //
;()result = $stmt->get_result$ //
;(outp = $result->fetch_all(MYSQLI_ASSOC$ //

;(echo json_encode($outp //
<? //

JSON HTML //

.JSON can very easily be translated into JavaScript //

.JavaScript can be used to make HTML in your web pages //

HTML Table //
:Make an HTML table with data received as JSON //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Make a table based on JSON data.</h2> //

 //

<script> //
;({const dbParam = JSON.stringify({table:"customers",limit:20 //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(const myObj = JSON.parse(this.responseText //
"<'let text = "<table border='1 //
} (for (let x in myObj //
;"<text += "<tr><td>" + myObj[x].name + "</td></tr //
{ //

```
"<text += "</table   //
;document.getElementById("demo").innerHTML = text   //
{ //
;("xmlhttp.open("POST", "json_demo_html_table.php //
;("xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded //
;(xmlhttp.send("x=" + dbParam //
<script/> //

<body/> //
<html/> //

Dynamic HTML Table //
:Make the HTML table based on the value of a drop down menu //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Make a table based on the value of a drop down menu.</h2> //

<"(select id="myselect" onchange="change_myselect(this.value> //
<option value="">Choose an option:</option>   //
<option value="customers">Customers</option>   //
<option value="products">Products</option>   //
<option value="suppliers">Suppliers</option>   //
<select/> //

<p id="demo"></p> //

<script> //
} (function change_myselect(sel //
;({const dbParam = JSON.stringify({table:sel,limit:20   //
;()const xmlhttp = new XMLHttpRequest   //
} ()xmlhttp.onload = function   //
;(myObj = JSON.parse(this.responseText     //
"<'text = "<table border='1     //
} (for (x in myObj     //
;"<text += "<tr><td>" + myObj[x].name + "</td></tr       //
{     //
"<text += "</table     //
;document.getElementById("demo").innerHTML = text     //
{   //
;(xmlhttp.open("POST", "json_demo_html_table.php", true   //
;("xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded   //
;(xmlhttp.send("x=" + dbParam   //
{ //
<script/> //
```

<body/> //
<html/> //

HTML Drop Down List //
:Make an HTML drop down list with data received as JSON //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Make a drop down list based on JSON data.</h2> //
 //

<script> //
;({const dbParam = JSON.stringify({table:"customers",limit:20 //
;()const xmlhttp = new XMLHttpRequest //
} ()xmlhttp.onload = function //
;(const myObj = JSON.parse(this.responseText //
"<let text = "<select //
} (for (let x in myObj //
;"<text += "<option>" + myObj[x].name + "</option //
{ //
"<text += "</select //
;document.getElementById("demo").innerHTML = text //
{ //
;("xmlhttp.open("POST", "json_demo_html_table.php //
;("xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded //
;(xmlhttp.send("x=" + dbParam //
<script/> //

<body/> //
<html/> //

JSONP //

.JSONP is a method for sending JSON data without worrying about cross-domain issues //

.JSONP does not use the XMLHttpRequest object //

.JSONP uses the <script> tag instead //

JSONP Intro //
.JSONP stands for JSON with Padding //

.Requesting a file from another domain can cause problems, due to cross-domain policy //

.Requesting an external script from another domain does not have this problem //

JSONP uses this advantage, and request files using the script tag instead of the //
.XMLHttpRequest object

<"script src="demo_jsonp.php> //
The Server File //
:The file on the server wraps the result inside a function call //

php?> //

;'{"myJSON = '{"name":"John", "age":30, "city":"New York$ //

;";(".echo "myFunc(".$myJSON //

<? //

The result returns a call to a function named "myFunc" with the JSON data as a //
.parameter

.Make sure that the function exists on the client //

The JavaScript function //
:The function named "myFunc" is located on the client, and ready to handle JSON data //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Request JSON using the script tag</h2> //
The PHP file returns a call to a function that will handle the JSON data. //
 //

<script> //
} (function myFunc(myObj //
;document.getElementById("demo").innerHTML = myObj.name //
{ //
<script/> //

<script src="demo_jsonp.php"></script> //

<body/> //
<html/> //

Creating a Dynamic Script Tag //
The example above will execute the "myFunc" function when the page is loading, based //
.on where you put the script tag, which is not very satisfying

:The script tag should only be created when needed //

```
<DOCTYPE html!> //
<html> //

<body> //

<h2>Click the Button.</h2> //
<p>A script tag with a src attribute is created and placed in the document.</p> //
<p>The PHP file returns a call to a function with the JSON object as a parameter.</p> //

<button onclick="clickButton()">Click me!</button> //

<p id="demo"></p> //

<script> //
} ()function clickButton //
;("let s = document.createElement("script   //
;"s.src = "demo_jsonp.php   //
;(document.body.appendChild(s   //
{ //

} (function myFunc(myObj //
;document.getElementById("demo").innerHTML = myObj.name   //
{ //
<script/> //

<body/> //
<html/> //

Dynamic JSONP Result //
.The examples above are still very static //

Make the example dynamic by sending JSON to the php file, and let the php file return a //
.JSON object based on the information it gets

PHP file //
php?> //
;("header("Content-Type: application/json; charset=UTF-8 //
;(obj = json_decode($_GET["x"], false$ //

;("conn = new mysqli("myServer", "myUser", "myPassword", "Northwind$ //
;(result = $conn->query("SELECT name FROM ".$obj->$table." LIMIT ".$obj->$limit$ //
;()outp = array$ //
;(outp = $result->fetch_all(MYSQLI_ASSOC$ //

;"(".(echo "myFunc(".json_encode($outp //
<? //
:PHP File explained //
.()Convert the request into an object, using the PHP function json_decode //
```

.Access the database, and fill an array with the requested data //
.Add the array to an object //
.Convert the array into JSON using the json_encode() function //
.Wrap "myFunc()" around the return object //

<DOCTYPE html!> //
<html> //
<body> //

A script tag with a src attribute is created and placed in the document. //
The PHP file returns a call to a function with an object as a parameter. //
 //

Try changing the table property from "customers" to "products". //

<script> //
;{ const obj = { table: "customers", limit: 10 //
;("let s = document.createElement("script //
;(s.src = "jsonp_demo_db.php?x=" + JSON.stringify(obj //
;(document.body.appendChild(s //

} (function myFunc(myObj //
;"" = let txt //
} (for (let x in myObj //
;"<txt += myObj[x].name + "<br //
{ //
;document.getElementById("demo").innerHTML = txt //
{ //
<script/> //

<body/> //
<html/> //

Callback Function //
When you have no control over the server file, how do you get the server file to call the //
?correct function

:Sometimes the server file offers a callback function as a parameter //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Request With a Callback Function</h2> //
The PHP file returns a call to the function you send as a callback. //
 //

<script> //

;("let s = document.createElement("script //
;"s.src = "demo_jsonp2.php?callback=myDisplayFunction //
;(document.body.appendChild(s //

} (function myDisplayFunction(myObj //
;document.getElementById("demo").innerHTML = myObj.name //
{ //
<script/> //

<body/> //
<html/> //

JavaScript / jQuery DOM Selectors //

jQuery vs JavaScript //
jQuery was created in 2006 by John Resig. It was designed to handle Browser //
Incompatibilities and to simplify HTML DOM Manipulation, Event Handling, Animations, and
.Ajax

.For more than 10 years, jQuery has been the most popular JavaScript library in the world //
However, after JavaScript Version 5 (2009), most of the jQuery utilities can be solved with //
:a few lines of standard JavaScript

Finding HTML Element by Id //
:"Return the element with id="id01 //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Finding HTML Elements by Id</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
} ()document).ready(function)$ //
;("var myElements = $("#id01 //
;(demo").text("The text from the id01 paragraph is: " + myElements[0].innerHTML#")$ //
;({ //
<script/> //

<body/> //

```
<html/> //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Finding HTML Elements by Id</h2> //
<p id="id01">Hello World!</p> //
<p id="id02">Hello Sweden!</p> //
<p id="id03">Hello Japan!</p> //

<p id="demo"></p> //

<script> //
} ()document).ready(function)$ //
;("var myElements = $("#id01    //
;(demo").text("The text from the id01 paragraph is: " + myElements[0].innerHTML#")$   //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Finding HTML Elements by Id</h2> //
<p id="id01">Hello World!</p> //
<p id="id02">Hello Sweden!</p> //
<p id="id03">Hello Japan!</p> //

<p id="demo"></p> //

<script> //
;("const myElement = document.getElementById("id01 //
document.getElementById("demo").innerHTML = "The text from the id01 paragraph is: " + //
;myElement.innerHTML
<script/> //

<body/> //
<html/> //
```

```
Finding HTML Elements by Tag Name //
:Return all <p> elements //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //

<body> //

<h2>Finding HTML Elements by Tag Name</h2> //

<p>Hello World!</p> //
<p>Hello Sweden!</p> //
<p>Hello Japan!</p> //

<p id="demo"></p> //

<script> //
} ()document).ready(function)$ //
;("var myElements = $("p    //
;(demo").text("The text in the first paragraph is: " + myElements[0].innerHTML#")$    //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Finding HTML Elements by Tag Name</h2> //
<p>Hello World!</p> //
<p>Hello Sweden!</p> //
<p>Hello Japan!</p> //

<p id="demo"></p> //

<script> //
;("const myElements = document.getElementsByTagName("p //
document.getElementById("demo").innerHTML = "The text in the first paragraph is: " + //
;myElements[0].innerHTML
<script/> //
```

<body/> //
<html/> //

Finding HTML Elements by Class Name //
."Return all elements with class="intro //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Finding HTML Elements by Class Name</h2> //

Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
} ()document).ready(function)$ //
;("var myElements = $(".intro //
;(demo").text("The first paragraph with class='intro' is: " + myElements[0].innerHTML#")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Finding HTML Elements by Class Name</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
;("const myElements = document.getElementsByClassName("intro //
document.getElementById("demo").innerHTML = "The first paragraph with class='intro' is: " //
;+ myElements[0].innerHTML

<script/> //

<body/> //
<html/> //

Finding HTML Elements by CSS Selectors //
."Return a list of all elements with class="intro //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Finding HTML Elements by Query Selector</h2> //

Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
} ()document).ready(function)$ //
;("var myElements = $("p.intro //
;(demo").text("The first paragraph with class='intro' is: " + myElements[0].innerHTML#")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Finding HTML Elements by Query Selector</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
;("const myElements = document.querySelectorAll("p.intro //

= document.getElementById("demo").innerHTML //
;The first paragraph with class='intro' is: " + myElements[0].innerHTML" //
<script/> //

<body/> //
<html/> //

JavaScript / jQuery HTML Elements //

jQuery vs JavaScript //
jQuery was created in 2006 by John Resig. It was designed to handle Browser //
Incompatibilities and to simplify HTML DOM Manipulation, Event Handling, Animations, and
.Ajax

.For more than 10 years, jQuery has been the most popular JavaScript library in the world //
However, after JavaScript Version 5 (2009), most of the jQuery utilities can be solved with //
:a few lines of standard JavaScript

Set Text Content //
:Set the inner text of an HTML element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Setting Text Content</h2> //
Hello World! //
Hello Sweden! //

<script> //
} ()document).ready(function)$ //
;("var myElement = $("#01 //
;("!myElement.text("Hello Sweden //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Setting Text Content</h2> //

Hello World! //
Hello Sweden! //

<script> //
;("const myElement = document.getElementById("01 //
;"!myElement.textContent = "Hello Sweden //
<script/> //

<body/> //
<html/> //

Get Text Content //
:Get the inner text of an HTML element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Get Text Content</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
} ()document).ready(function)$ //
;()var myText = $("#02").text //
;(demo").text(myText#")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get Text Content</h2> //
Hello World! //

```
<p id="02">Hello Sweden!</p> //
<p id="03">Hello Japan!</p> //

<p id="demo"></p> //

<script> //
;const myText = document.getElementById("02").textContent //
;document.getElementById("demo").innerHTML = myText //
<script/> //

<body/> //
<html/> //

Set HTML Content //
:Set the HTML content of an element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Set HTML</h2> //
<div id="01"><p>Hello World!</p></div> //
<div id="02"><p>Hello Sweden!</p></div> //

<p id="demo"></p> //

<script> //
} ()document).ready(function)$ //
;("<html("<p>Hello World!</p.("#02")$   //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Set HTML</h2> //
<div id="01"><p>Hello World!</p></div> //
<div id="02"><p>Hello Sweden!</p></div> //
```

```
<p id="demo"></p> //

<script> //
;"<document.getElementById("02").innerHTML = "<p>Hello World!</p //
<script/> //

<body/> //
<html/> //
```

Get HTML Content //
:Get the HTML content of an element //

```
<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Set HTML</h2> //

<div id="01"><p>Hello World!</p></div> //
<div id="02"><p>Hello Sweden!</p></div> //

<script> //
} ()document).ready(function)$ //
;()var content = $("#02").html   //
;(html(content.("#01")$   //
;({ //
<script/> //

<body/> //
<html/> //
```

:or //

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>Setting HTML</h2> //

<div id="01"><p>Hello World!</p></div> //
<div id="02"><p>Hello Sweden!</p></div> //

<script> //
;const content = document.getElementById("02").innerHTML //
;document.getElementById("01").innerHTML = content //
```

<script/> //

<body/> //
<html/> //

JavaScript / jQuery CSS Styles //

jQuery vs JavaScript //
jQuery was created in 2006 by John Resig. It was designed to handle Browser //
Incompatibilities and to simplify HTML DOM Manipulation, Event Handling, Animations, and
.Ajax

.For more than 10 years, jQuery has been the most popular JavaScript library in the world //
However, after JavaScript Version 5 (2009), most of the jQuery utilities can be solved with //
:a few lines of standard JavaScript

Hiding HTML Elements //
:Hide an HTML Element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Get Text Content</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

 //

<script> //
} ()document).ready(function)$ //
;()hide.("#02")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Hide HTML Elements</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

<script> //
;"document.getElementById("02").style.display = "none //
<script/> //

<body/> //
<html/> //

Showing HTML Elements //
:Show an HTML Element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Show HTML Elements</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

<script> //
} ()document).ready(function)$ //
;()show.("#02")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Show HTML Elements</h2> //
Hello World! //
Hello Sweden! //
Hello Japan! //

<script> //
;"" = document.getElementById("02").style.display //
<script/> //

<body/> //
<html/> //

Styling HTML Elements //
:Change the font size of an HTML element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

Change the style of an HTML element. //

<script> //
} ()document).ready(function)$ //
;("demo").css("font-size","35px#")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

Change the style of an HTML element. //

<script> //
;"document.getElementById("demo").style.fontSize = "35px //
<script/> //

<body/> //
<html/> //

JavaScript / jQuery HTML DOM //

jQuery vs JavaScript //

jQuery was created in 2006 by John Resig. It was designed to handle Browser // Incompatibilities and to simplify HTML DOM Manipulation, Event Handling, Animations, and .Ajax

.For more than 10 years, jQuery has been the most popular JavaScript library in the world // However, after JavaScript Version 5 (2009), most of the jQuery utilities can be solved with // :a few lines of standard JavaScript

Removing HTML Elements //
:Remove an HTML element //

```
<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Remove an HTML Element</h2> //
<p id="id01">Hello World!</p> //
<p id="id02">Hello Sweden!</p> //

<script> //
} ()document).ready(function)$ //
;()id02").remove#")$   //
;({ //
<script/> //

<body/> //
<html/> //
```

:or //

```
<DOCTYPE html!> //
<html> //
<body> //

<h2>Remove an HTML Element</h2> //
<p id="id01">Hello World!</p> //
<p id="id02">Hello Sweden!</p> //

<script> //
;()document.getElementById("id02").remove //
<script/> //

<body/> //
<html/> //
```

Get Parent Element //
:Return the parent of an HTML element //

<DOCTYPE html!> //
<html> //
<head> //
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script> //
<head/> //
<body> //

<h2>Getting Parent HTML Element</h2> //
Hello World! //
Hello Sweden! //

 //

<script> //
} ()document).ready(function)$ //
;(("demo").text($("#02").parent().prop("nodeName#")$ //
;({ //
<script/> //

<body/> //
<html/> //

:or //

<DOCTYPE html!> //
<html> //
<body> //

<h2>Get Parent HTML Element</h2> //
Hello World! //
Hello Sweden! //

 //

<script> //
document.getElementById("demo").innerHTML = //
;document.getElementById("02").parentNode.nodeName
<script/> //

<body/> //
<html/> //

JavaScript Graphics //

Graphic Libraries //

:JavaScript libraries to use for both Artificial Intelligence graphs and other charts //

Plotly.js //
Chart.js //
Google Chart //
Plotly.js //

Chart.js //
:Chart.js comes with many built-in chart types //

Scatter //
Line //
Bar //
Radar //
Pie and Doughnut //
Polar Area //
Bubble //

Google Chart //
From simple line charts to complex tree maps, Google Chart provides a number of built-in //
:chart types

Scatter Chart //
Line Chart //
Bar / Column Chart //
Area Chart //
Pie Chart //
Donut Chart //
Org Chart //
Map / Geo Chart //

HTML Canvas //

HTML Canvas is perfect for Scatter Plots //

HTML Canvas is perfect for Line Graphs //

HTML Canvas is perfect for combining Scatter and Lines //

Scatter Plots //

<DOCTYPE html!> //
<html> //
<body> //

canvas id="myCanvas" width="400" height="400" style="border:1px solid> //
<grey"></canvas

```
<script> //
;("const canvas = document.getElementById("myCanvas //
;("const ctx = canvas.getContext("2d //
;canvas.height = canvas.width //
(ctx.transform(1, 0, 0, -1, 0, canvas.height //

;[const xArray = [50,60,70,80,90,100,110,120,130,140,150 //
;[const yArray = [7,8,8,9,9,9,10,11,14,14,15 //

;"ctx.fillStyle = "red //
} (++for (let i = 0; i < xArray.length-1; i //
;let x = xArray[i]*400/150   //
;let y = yArray[i]*400/15   //
;()ctx.beginPath   //
;(ctx.ellipse(x, y, 3, 3, 0, 0, Math.PI * 2   //
;()ctx.fill   //
{ //
<script/> //

<body/> //
<html/> //
```

Line Graphs //

```
<DOCTYPE html!> //
<html> //
<body> //
canvas id="myCanvas" width="400" height="400" style="border:1px solid> //
<grey"></canvas

<script> //
;("const canvas = document.getElementById("myCanvas //
;("const ctx = canvas.getContext("2d //
;"ctx.fillStyle = "#FF0000 //
;canvas.height = canvas.width //
(ctx.transform(1, 0, 0, -1, 0, canvas.height //

;let xMax = canvas.height //
;let slope = 1.2 //
;let intercept = 70 //

;(ctx.moveTo(0, intercept //
;((ctx.lineTo(xMax, f(xMax //
;"ctx.strokeStyle = "black //
;()ctx.stroke //

} (function f(x //
;return x * slope + intercept   //
```

```
{ //
<script/> //

<body/> //
<html/> //

Combined //

<DOCTYPE html!> //
<html> //
<body> //
canvas id="myCanvas" width="400" height="400" style="border:1px solid> //
<grey"></canvas

<script> //
;("const canvas = document.getElementById("myCanvas //
;("const ctx = canvas.getContext("2d //
;"ctx.fillStyle = "#FF0000 //
;canvas.height = canvas.width //
(ctx.transform(1, 0, 0, -1, 0, canvas.height //

;let xMax = canvas.height //
;let yMax = canvas.width //
;let slope = 1.2 //
;let intercept = 70 //

;[const xArray = [50,60,70,80,90,100,110,120,130,140,150 //
;[const yArray = [7,8,8,9,9,9,10,11,14,14,15 //

Plot Scatter // //
;"ctx.fillStyle = "red //
} (++for (let i = 0; i < xArray.length-1; i //
;let x = xArray[i]*xMax/150   //
;let y = yArray[i]*yMax/15   //
;()ctx.beginPath   //
;(ctx.ellipse(x, y, 3, 3, 0, 0, Math.PI * 2   //
;()ctx.fill   //
{ //

Plot Line // //
;(ctx.moveTo(0, intercept //
;((ctx.lineTo(xMax, f(xMax //
;"ctx.strokeStyle = "black //
;()ctx.stroke //

<Line Function<br // //
} (function f(x //
;return x * slope + intercept   //
```

```
{ //
<script/> //

<body/> //
<html/> //


Plotly.js //

Plotly.js is a charting library that comes with over 40 chart types, 3D charts, statistical //
.graphs, and SVG maps

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;[var xArray = [50,60,70,80,90,100,110,120,130,140,150 //
;[var yArray = [7,8,8,9,9,9,10,11,14,14,15 //

Define Data // //
}] = var data //
,x:xArray   //
,y:yArray   //
"mode:"markers   //
;[{ //

Define Layout // //
} = var layout //
,{"xaxis: {range: [40, 160], title: "Square Meters   //
,{"yaxis: {range: [5, 16], title: "Price in Millions   //
"title: "House Prices vs. Size   //
;{ //

Display using Plotly // //
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Line Graphs //

<DOCTYPE html!> //
<html> //
```

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;[var xArray = [50,60,70,80,90,100,110,120,130,140,150 //
;[var yArray = [7,8,8,9,9,9,10,11,14,14,15 //

Define Data // //
}] = var data //
,x: xArray   //
,y: yArray   //
"mode:"lines   //
;[{ //

Define Layout // //
} = var layout //
,{"xaxis: {range: [40, 160], title: "Square Meters   //
,{"yaxis: {range: [5, 16], title: "Price in Millions   //
"title: "House Prices vs. Size   //
;{ //

Display using Plotly // //
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Linear Graphs //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //

<body> //
<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;"var exp = "x + 17 //

Generate values // //
;[] = var xValues //
;[] = var yValues //
} (for (var x = 0; x <= 10; x += 1 //
;(xValues.push(x   //
;((yValues.push(eval(exp   //
```

```
{ //

Define Data // //
}] = var data //
,x: xValues   //
,y: yValues   //
"mode:"lines   //
;[{ //

Define Layout // //
;{var layout = {title: "y = " + exp //

Display using Plotly // //
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Multiple Lines //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //

<body> //
<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;"var exp1 = "x //
;"var exp2 = "1.5*x //
;"var exp3 = "1.5*x + 7 //
Generate values // //

;[] = var x1Values //
;[] = var x2Values //
;[] = var x3Values //
;[] = var y1Values //
;[] = var y2Values //
;[] = var y3Values //

} (for (var x = 0; x <= 10; x += 1 //
;(x1Values.push(x   //
;(x2Values.push(x   //
;(x3Values.push(x   //
;((y1Values.push(eval(exp1   //
;((y2Values.push(eval(exp2   //
;((y3Values.push(eval(exp3   //
```

```
{ //

Define Data // //
] = var data //
,{"x: x1Values, y: y1Values, mode:"lines}   //
,{"x: x2Values, y: y2Values, mode:"lines}   //
{"x: x3Values, y: y3Values, mode:"lines}   //
;[ //

Define Layout// //
;{"[" + var layout = {title: "[y=" + exp1 + "]  [y=" + exp2 + "]  [y=" + exp3 //

Display using Plotly // //
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Bar Charts //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;["var xArray = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yArray = [55, 49, 44, 24, 15 //

}] = var data //
,x:xArray   //
,y:yArray   //
"type:"bar   //
;[{ //

;{"var layout = {title:"World Wide Wine Production //

;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Horizontal Bar Charts //
```

```
<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;[var xArray = [55, 49, 44, 24, 15 //
;[" var yArray = ["Italy ", "France ", "Spain ", "USA ", "Argentina //

}] = var data //
,x:xArray   //
,y:yArray   //
,"type:"bar   //
,"orientation:"h   //
{"(marker: {color:"rgba(255,0,0,0.6   //
;[{ //

;{"var layout = {title:"World Wide Wine Production //

;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //
```

Pie Charts //

```
<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;["var xArray = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yArray = [55, 49, 44, 24, 15 //

;{"var layout = {title:"World Wide Wine Production //

;[{"var data = [{labels:xArray, values:yArray, type:"pie //

;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
```

```
<html/> //

Donut Charts //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //
<body> //

<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;["var xArray = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yArray = [55, 49, 44, 24, 15 //

;{"var layout = {title:"World Wide Wine Production //

;[{"var data = [{labels:xArray, values:yArray, hole:.4, type:"pie //

;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

Plotting Equations //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //

<body> //
<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;"(var exp = "Math.sin(x //

Generate values // //
;[] = var xValues //
;[] = var yValues //
} (for (var x = 0; x <= 10; x += 0.1 //
;(xValues.push(x   //
;((yValues.push(eval(exp   //
{ //

Display using Plotly // //
;[{"var data = [{x:xValues, y:yValues, mode:"lines //
;{var layout = {title: "y = " + exp //
```

```
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

<body/> //
<html/> //

<DOCTYPE html!> //
<html> //
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script> //

<body> //
<div id="myPlot" style="width:100%;max-width:700px"></div> //

<script> //
;"(var exp = "Math.cos(x //

Generate values // //
;[] = var xValues //
;[] = var yValues //
} (for (var x = 0; x <= 10; x += 0.2 //
;((yValues.push(eval(exp   //
;(xValues.push(x   //
{ //

Display using Plotly // //
;[{"var data = [{x:xValues, y:yValues, mode:"markers //
;{var layout = {title: "y = " + exp //
;(Plotly.newPlot("myPlot", data, layout //
<script/> //

Chart.js //

Chart.js is an free JavaScript library for making HTML-based charts. It is one of the //
simplest visualization libraries for JavaScript, and comes with the following built-in chart
:types

Scatter Plot //
Line Chart //
Bar Chart //
Pie Chart //
Donut Chart //
Bubble Chart //
Area Chart //
Radar Chart //
Mixed Chart //
?How to Use Chart.js //
.Chart.js is easy to use //
```

:(First, add a link to the providing CDN (Content Delivery Network //

script> //
<"src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js //
<script/> //
:Then, add a <canvas> to where you want to draw the chart //

<canvas id="myChart" style="width:100%;max-width:700px"></canvas> //
.The canvas element must have a unique id //

!That's all //

:Typical Scatter Chart Syntax //
} ,"const myChart = new Chart("myChart //
,"type: "scatter //
,{} :data //
{} :options //
;({ //
:Typical Line Chart Syntax //
} ,"const myChart = new Chart("myChart //
,"type: "line //
,{} :data //
{} :options //
;({ //
:Typical Bar Chart Syntax //
} ,"const myChart = new Chart("myChart //
,"type: "bar //
,{} :data //
{} :options //
;({ //
Scatter Plots //
House Prices vs. Size //

<DOCTYPE html!> //
<html> */}
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script>
<body>
<canvas id="myChart" style="width:100%;max-width:700px"></canvas>

<script>
] = var xyValues
,{x:50, y:7}
,{x:60, y:8}
,{x:70, y:8}
,{x:80, y:9}
,{x:90, y:9}
,{x:100, y:9}
,{x:110, y:10}

```
,{x:120, y:11}
,{x:130, y:14}
,{x:140, y:14}
{x:150, y:15}
;[

} ,"new Chart("myChart
,"type: "scatter
} :data
}] :datasets
,pointRadius: 4
,"(pointBackgroundColor: "rgb(0,0,255
data: xyValues
[{
,{
} :options
,{legend: {display: false
} :scales
,[{{xAxes: [{ticks: {min: 40, max:160
,[{{yAxes: [{ticks: {min: 6, max:16
{
{
;({
<script/>

<body/>
<html/>

{/*

Line Graphs  //
House Prices vs. Size  //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;[const xValues = [50,60,70,80,90,100,110,120,130,140,150 //
;[const yValues = [7,8,8,9,9,9,10,11,14,14,15 //

} ,"new Chart("myChart //
,"type: "line   //
} :data   //
,labels: xValues     //
}] :datasets     //
```

```
,fill: false       //
,lineTension: 0       //
,"(backgroundColor: "rgba(0,0,255,1.0       //
,"(borderColor: "rgba(0,0,255,0.1       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :scales     //
,[{{yAxes: [{ticks: {min: 6, max:16       //
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //

:If you set the borderColor to zero, you can scatter plot the line graph //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;[var xValues = [50,60,70,80,90,100,110,120,130,140,150 //
;[var yValues = [7,8,8,9,9,9,10,11,14,14,15 //

} ,"new Chart("myChart //
,"type: "line   //
} :data   //
,labels: xValues     //
}] :datasets     //
,fill: false       //
,lineTension: 0       //
,"(backgroundColor: "rgba(0,0,255,1       //
,"(borderColor: "rgba(0,0,0,0       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :scales     //
,[{{yAxes: [{ticks: {min: 6, max:16       //
{     //
```

```
{   //
;({ //
<script/> //

<body/> //
<html/> //
```

Multiple Lines //

```
<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script> //
<body> //
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;[const xValues = [100,200,300,400,500,600,700,800,900,1000 //

} ,"new Chart("myChart //
,"type: "line   //
} :data   //
,labels: xValues     //
}] :datasets     //
,[data: [860,1140,1060,1060,1070,1110,1330,2210,7830,2478       //
,"borderColor: "red       //
fill: false       //
} ,{     //
,[data: [1600,1700,1700,1900,2000,2700,4000,5000,6000,7000       //
,"borderColor: "green       //
fill: false       //
} ,{     //
,[data: [300,700,2000,5000,6000,4000,2000,1000,200,100       //
,"borderColor: "blue       //
fill: false       //
[{     //
,{   //
} :options   //
{legend: {display: false     //
{   //
;({ //
<script/> //
```

Linear Graphs //

```
<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script> //
<body> //
```

```
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;[] = const xValues //
;[] = const yValues //
;(generateData("x * 2 + 7", 0, 10, 0.5 //

} ,"new Chart("myChart //
,"type: "line   //
} :data   //
,labels: xValues     //
}] :datasets     //
,fill: false       //
,pointRadius: 1       //
,"(borderColor: "rgba(255,0,0,0.5       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :title     //
,display: true       //
,"text: "y = x * 2 + 7       //
fontSize: 16       //
{     //
{   //
;({ //
} (function generateData(value, i1, i2, step = 1 //
} (for (let x = i1; x <= i2; x += step   //
;((yValues.push(eval(value     //
;(xValues.push(x     //
{   //
{ //
<script/> //

<body/> //
<html/> //

Function Graphs //

:(Same as Linear Graph. Just change the generateData parameter(s //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script> //
<body> //
```

```
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;[] = var xValues //
;[] = var yValues //
;(generateData("Math.sin(x)", 0, 10, 0.5 //

} ,"new Chart("myChart //
,"type: "line   //
} :data   //
,labels: xValues     //
}] :datasets     //
,fill: false       //
,pointRadius: 2       //
,"(borderColor: "rgba(0,0,255,0.5       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :title     //
,display: true       //
,"(text: "y = sin(x       //
fontSize: 16       //
{     //
{   //
;({ //
} (function generateData(value, i1, i2, step = 1 //
} (for (let x = i1; x <= i2; x += step   //
;((yValues.push(eval(value     //
;(xValues.push(x     //
{   //
{ //
<script/> //

<body/> //
<html/> //

Bar Charts //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
```

```
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
;["var barColors = ["red", "green","blue","orange","brown //

} ,"new Chart("myChart //
,"type: "bar   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :title     //
,display: true       //
"text: "World Wine Production 2018       //
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //

:Color only one bar //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
;["var barColors = ["blue //

} ,"new Chart("myChart //
,"type: "bar   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
```

```
,{   //
} :options   //
,{legend: {display: false     //
} :scales     //
}] :yAxes       //
} :ticks         //
beginAtZero: true           //
{          //
,[{       //
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //

:Same color all bars //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
;"var barColors = "red //

} ,"new Chart("myChart //
,"type: "bar   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :scales     //
}] :yAxes       //
} :ticks         //
beginAtZero: true           //
{          //
,[{       //
```

```
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //

:Color Shades //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
] = var barColors //
,"(rgba(255,0,0,1.0"   //
,"(rgba(255,0,0,0.8"   //
,"(rgba(255,0,0,0.6"   //
,"(rgba(255,0,0,0.4"   //
"(rgba(255,0,0,0.2"   //
;[ //

} ,"new Chart("myChart //
,"type: "bar   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
,{   //
} :options   //
,{legend: {display: false     //
} :scales     //
}] :yAxes       //
} :ticks         //
beginAtZero: true           //
{         //
,[{       //
{     //
{   //
;({ //
```

<script/> //

<body/> //
<html/> //

Horizontal Bars //
:"Just change type from "bar" to "horizontalBar //

<DOCTYPE html!> //
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
;["var barColors = ["red", "green","blue","orange","brown //

} ,"new Chart("myChart //
,"type: "horizontalBar //
} :data //
,labels: xValues //
}] :datasets //
,backgroundColor: barColors //
data: yValues //
[{ //
,{ //
} :options //
,{legend: {display: false //
} :title //
,display: true //
"text: "World Wine Production 2018 //
,{ //
} :scales //
[{{xAxes: [{ticks: {min: 10, max:60 //
{ //
{ //
;({ //
<script/> //

<body/> //
<html/> //

Pie Charts //

<DOCTYPE html!> //

```
<html> //
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script> //
<body> //

<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
] = var barColors //
,"b91d47#"   //
,"#00aba9"   //
,"#2b5797"   //
,"e8c3b9#"   //
"#1e7145"   //
;[ //

} ,"new Chart("myChart //
,"type: "pie   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
,{   //
} :options   //
} :title     //
,display: true       //
"text: "World Wide Wine Production 2018       //
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //
```

```
:"Just change type from "pie" to "doughnut //

<DOCTYPE html!> //
<html> //
script> //
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></
<script
<body> //
```

```
<canvas id="myChart" style="width:100%;max-width:600px"></canvas> //

<script> //
;["var xValues = ["Italy", "France", "Spain", "USA", "Argentina //
;[var yValues = [55, 49, 44, 24, 15 //
] = var barColors //
,"b91d47#"   //
,"#00aba9"   //
,"#2b5797"   //
,"e8c3b9#"   //
"#1e7145"   //
;[ //

} ,"new Chart("myChart //
,"type: "doughnut   //
} :data   //
,labels: xValues     //
}] :datasets     //
,backgroundColor: barColors       //
data: yValues       //
[{     //
,{   //
} :options   //
} :title     //
,display: true       //
"text: "World Wide Wine Production 2018       //
{     //
{   //
;({ //
<script/> //

<body/> //
<html/> //

Google Chart //

From simple line charts to complex hierarchical tree maps, the //
Google Chart gallery provides a large number of ready-to-use chart
:types

Scatter Chart //
Line Chart //
Bar / Column Chart //
```

```
Area Chart //
Pie Chart //
Donut Chart //
Org Chart //
Map / Geo Chart //
?How to Use Google Chart //
To use Google Chart in your web page, add a link to the charts //
:loader

script> //
<"src="https://www.gstatic.com/charts/loader.js //
<script/> //
.Google Chart is easy to use //

:Just add a <div> element to display the chart //

<div id="myChart" style="max-width:700px; height:400px"></div> //
.The <div> element must have a unique id //

:Then load the Google Graph API //

Load the Visualization API and the corechart package //
Set a callback function to call when the API is loaded //
;({['google.charts.load('current',{packages:['corechart 1 //

;(google.charts.setOnLoadCallback(drawChart 2 //
!That's all //

Line Graph //

<DOCTYPE html!> //
<html> //
script type="text/javascript"> //
<src="https://www.gstatic.com/charts/loader.js"></script
<body> //
div id="myChart" style="width:100%; max-width:600px;> //
<height:500px;"></div

<script> //
;({['google.charts.load('current',{packages:['corechart //
;(google.charts.setOnLoadCallback(drawChart //

} ()function drawChart //
```

```
Set Data // //
])var data = google.visualization.arrayToDataTable //
,['Price', 'Size']   //
,[90,9],[80,9],[70,8],[60,8],[50,7]   //
,[120,11],[110,10],[100,9]   //
[150,15],[140,14],[130,14]   //
;([ //
Set Options // //
} = var options //
,'title: 'House Prices vs. Size   //
,{'hAxis: {title: 'Square Meters   //
,{'vAxis: {title: 'Price in Millions   //
'legend: 'none   //
;{ //
Draw // //
var chart = new //
;(('google.visualization.LineChart(document.getElementById('myChart
;(chart.draw(data, options //
{ //
<script/> //

<body/> //
<html/> //

Scatter Plots //
To scatter plot the same data, change google.visualization to //
:ScatterChart

<DOCTYPE html!> //
<html> //
script type="text/javascript"> //
<src="https://www.gstatic.com/charts/loader.js"></script
<body> //
div id="myChart" style="width:100%; max-width:600px;> //
<height:500px;"></div

<script> //
;({['google.charts.load('current',{packages:['corechart //
;(google.charts.setOnLoadCallback(drawChart //

} ()function drawChart //
Set Data // //
])var data = google.visualization.arrayToDataTable //
```

```
,['Price', 'Size']   //
,[90,9],[80,9],[70,8],[60,8],[50,7]   //
,[120,11],[110,10],[100,9]   //
[150,15],[140,14],[130,14]   //
;([ //
Set Options // //
} = var options //
,'title: 'House Prices vs. Size   //
,{'hAxis: {title: 'Square Meters   //
,{'vAxis: {title: 'Price in Millions   //
'legend: 'none   //
;{ //
Draw // //
var chart = new //
;(('google.visualization.ScatterChart(document.getElementById('myChart
;(chart.draw(data, options //
{ //
<script/> //

<body/> //
<html/> //

Bar Charts //

<DOCTYPE html!> //
<html> //
script type="text/javascript"> //
<src="https://www.gstatic.com/charts/loader.js"></script

<body> //
div id="myChart" style="width:100%; max-width:600px;> //
<height:500px;"></div

<script> //
;({['google.charts.load('current', {'packages':['corechart //
;(google.charts.setOnLoadCallback(drawChart //

} ()function drawChart //
])var data = google.visualization.arrayToDataTable //
,['Contry', 'Mhl']   //
,[Italy',55']   //
,[France',49']   //
,[Spain',44']   //
```

```
,[USA',24']   //
[Argentina',15']   //
;([ //

} = var options //
'title:'World Wide Wine Production   //
;{ //

var chart = new //
;(('google.visualization.BarChart(document.getElementById('myChart
;(chart.draw(data, options   //
{ //
<script/> //

<body/> //
<html/> //

<DOCTYPE html!> //
<html> //
script type="text/javascript"> //
<src="https://www.gstatic.com/charts/loader.js"></script

<body> //
div> //
<";id="myChart" style="width:100%; max-width:600px; height:500px //
<div/> //

<script> //
;({['google.charts.load('current', {'packages':['corechart //
;(google.charts.setOnLoadCallback(drawChart //

} ()function drawChart //
])var data = google.visualization.arrayToDataTable //
,['Contry', 'Mhl']   //
,[Italy',54.8']   //
,[France',48.6']   //
,[Spain',44.4']   //
,[USA',23.9']   //
[Argentina',14.5']   //
;([ //

} = var options //
'title:'World Wide Wine Production   //
```

```
;{ //

var chart = new //
;(('google.visualization.PieChart(document.getElementById('myChart
;(chart.draw(data, options   //
{ //
<script/> //

<body/> //
<html/> //

3D Pie //
:To display the Pie in 3D, just add is3D: true to the options //

<DOCTYPE html!> //
<html> //
script type="text/javascript"> //
<src="https://www.gstatic.com/charts/loader.js"></script

<body> //
div> //
<";id="myChart" style="width:100%; max-width:600px; height:500px //
<div/> //

<script> //
;({['google.charts.load('current', {'packages':['corechart //
;(google.charts.setOnLoadCallback(drawChart //

} ()function drawChart //
])var data = google.visualization.arrayToDataTable //
,['Contry', 'Mhl']   //
,[Italy',54.8']   //
,[France',48.6']   //
,[Spain',44.4']   //
,[USA',23.9']   //
[Argentina',14.5']   //
;([ //

} = var options //
,'title:'World Wide Wine Production   //
is3D:true   //
;{ //
```

```
var chart = new //
;(('google.visualization.PieChart(document.getElementById('myChart
;(chart.draw(data, options   //
{ //
<script/> //

<body/> //
<html/> //

D3.js //

.D3.js is a JavaScript library for manipulating HTML data //

.D3.js is easy to use //

?How to Use D3.js //
:To use D3.js in your web page, add a link to the library //

<script src="//d3js.org/d3.v3.min.js"></script> //
This script selects the body element and appends a paragraph with //
:"!the text "Hello World

<DOCTYPE html!> //
<html> //
<script src="//d3js.org/d3.v4.js"></script> //
<body> //
<h2>D3.js is Easy to Use</h2> //
p>The script below selects the body element and appends a paragraph> //
<with the text "Hello World!":</p

<script> //
;("!d3.select("body").append("p").text("Hello World //
<script/> //

<body/> //
<html/> //

Scatter Plot //

<DOCTYPE html!> //
<html> //
<script src="https://d3js.org/d3.v4.js"></script> //
<body> //
```

```
<h2>D3.js Scatter-Plot</h2> //

<svg id="myPlot" style="width:500px;height:500px"></svg> //

<script> //
Set Dimensions // //
;const xSize = 500 //
;const ySize = 500 //
;const margin = 40 //
;const xMax = xSize - margin*2 //
;const yMax = ySize - margin*2 //

Create Random Points // //
;const numPoints = 100 //
;[] = const data //
} (++for (let i = 0; i < numPoints; i //
;([data.push([Math.random() * xMax, Math.random() * yMax   //
{ //

Append SVG Object to the Page // //
("const svg = d3.select("#myPlot //
("append("svg.   //
("append("g.   //
;("(" + attr("transform","translate(" + margin + "," + margin.   //

X Axis // //
()const x = d3.scaleLinear //
([domain([0, 500.   //
;([range([0, xMax.   //

("svg.append("g //
("(" + attr("transform", "translate(0," + yMax.   //
;((call(d3.axisBottom(x.   //

Y Axis // //
()const y = d3.scaleLinear //
([domain([0, 500.   //
;([range([ yMax, 0.   //

("svg.append("g //
;((call(d3.axisLeft(y.   //

Dots // //
```

```
('svg.append('g //
("selectAll("dot.   //
()data(data).enter.   //
("append("circle.   //
( { [attr("cx", function (d) { return d[0.   //
( { [attr("cy", function (d) { return d[1.   //
(attr("r", 3.   //
;("style("fill", "Red.   //
<script/> //

<body/> //
<html/> //

JavaScript Reference //

Property/Method  Description Belongs To //
abs()  Returns the absolute value of x Math //
accessKey  Sets or returns the accesskey attribute of an element //
Element
acos() Returns the arccosine of x, in radians  Math //
acosh()  Returns the hyperbolic arccosine of x Math //
activeElement  Returns the currently focused element in the document //
Document
addEventListener() Attaches an event handler to the document //
Document, Element
adoptNode()  Adopts a node from another document Document //
alert()  Displays an alert box with a message and an OK button //
Window
altKey Returns whether the "ALT" key was pressed when the mouse //
event was triggered  MouseEvent, KeyboardEvent, TouchEvent
anchors  Returns a collection of all <a> elements in the document //
that have a name attribute Document
animationName  Returns the name of the animation AnimationEvent //
appCodeName  Returns the code name of the browser  Navigator //
appendChild()  Adds a new child node, to an element, as the last //
child node  Element
applets  Returns a collection of all <applet> elements in the //
document Document
appName  Returns the name of the browser Navigator //
appVersion Returns the version information of the browser  Navigator //
asin() Returns the arcsine of x, in radians  Math //
asinh()  Returns the hyperbolic arcsine of x Math //
```

```
assert() Writes an error message to the console if the assertion is //
false  Console
assign() Loads a new document  Location //
atan() Returns the arctangent of x as a numeric value between -PI/2 //
and PI/2 radians Math
atan2()  Returns the arctangent of the quotient of its arguments //
Math
atanh()  Returns the hyperbolic arctangent of x  Math //
atob() Decodes a base-64 encoded string  Window //
attributes Returns a NamedNodeMap of an element's attributes Element //
availHeight  Returns the height of the screen (excluding the Windows //
Taskbar)  Screen
availWidth Returns the width of the screen (excluding the Windows //
Taskbar) Screen
back() Loads the previous URL in the history list  History //
baseURI  Returns the absolute base URI of a document Document //
blur() Removes focus from an element Element, Window //
body Sets or returns the document's body (the <body> element) //
Document
break  Exits a switch or a loop  Statements //
btoa() Encodes a string in base-64 Window //
bubbles  Returns whether or not a specific event is a bubbling event //
Event
button Returns which mouse button was pressed when the mouse event //
was triggered MouseEvent
buttons  Returns which mouse buttons were pressed when the mouse //
event was triggered MouseEvent
cancelable Returns whether or not an event can have its default //
action prevented Event
cbrt() Returns the cubic root of x Math //
ceil() Returns x, rounded upwards to the nearest integer Math //
changeTouches  Returns a list of all the touch objects whose state //
changed between the previous touch and this touch TouchEvent
characterSet Returns the character encoding for the document //
Document
charAt() Returns the character at the specified index (position) //
String
charCode Returns the Unicode character code of the key that //
triggered the onkeypress event KeyboardEvent
charCodeAt() Returns the Unicode of the character at the specified //
index String
charset  Deprecated. Use characterSet instead. Returns the character //
encoding for the document Document
```

```
childElementCount  Returns the number of child elements an element //
has Element
childNodes Returns a collection of an element's child nodes //
(including text and comment nodes) Element
children Returns a collection of an element's child element //
(excluding text and comment nodes) Element
classList  Returns the class name(s) of an element Element //
class  Declares a class  Statements //
className  Sets or returns the value of the class attribute of an //
element  Element
clear()  Clears the console  Console, Storage //
clearInterval()  Clears a timer set with setInterval() Window //
clearTimeout() Clears a timer set with setTimeout()  Window //
clearWatch() Unregister location/error monitoring handlers //
previously installed using Geolocation.watchPosition()  Geolocation
click()  Simulates a mouse-click on an element Element //
clientHeight Returns the height of an element, including padding //
Element
clientLeft Returns the width of the left border of an element //
Element
clientTop  Returns the width of the top border of an element Element //
clientWidth  Returns the width of an element, including padding //
Element
clientX  Returns the horizontal coordinate of the mouse pointer, //
relative to the current window, when the mouse event was triggered
MouseEvent, TouchEvent
clientY  Returns the vertical coordinate of the mouse pointer, //
relative to the current window, when the mouse event was triggered
MouseEvent, TouchEvent
clipboardData  Returns an object containing the data affected by the //
clipboard operation ClipboardData
closed Returns a Boolean value indicating whether a window has been //
closed or not  Window
close()  Closes the output stream previously opened with //
document.open() Document, Window
closest()  Searches up the DOM tree for the closest element which //
matches a specified CSS selector Element
clz32(x) Returns the number of leading zeros in a 32-bit binary //
representation of x  Math
code Returns the code of the key that triggered the event //
KeyboardEvent
colorDepth Returns the bit depth of the color palette for displaying //
images  Screen
```

```
compareDocumentPosition()  Compares the document position of two //
elements  Element
compile()  Deprecated in version 1.5. Compiles a regular expression //
RegExp
composed Returns whether the event is composed or not  Event //
concat() Joins two or more arrays, and returns a copy of the joined //
arrays Array, String
confirm()  Displays a dialog box with a message and an OK and a //
Cancel button  Window
const  Declares a variable with a constant value Statements //
constructor()  Creates and initialize objects created within a class //
Classes
constructor  Returns the function that created the Array object's //
prototype  Array, Boolean, Date, Number, RegExp
contains() Returns true if a node is a descendant of a node, //
otherwise false Element
contentEditable  Sets or returns whether the content of an element //
is editable or not  Element
continue Breaks one iteration (in the loop) if a specified condition //
occurs, and continues with the next iteration in the loop Statements
console  Returns a reference to the Console object, which provides //
methods for logging information to the browser's console (See Console
object) Window
cookie Returns all name/value pairs of cookies in the document //
Document
cookieEnabled  Determines whether cookies are enabled in the browser //
Navigator
coordinates  Returns the position and altitude of the device on //
Earth  Geolocation
copyWithin() Copies array elements within the array, to and from //
specified positions Array
cos(x) Returns the cosine of x (x is in radians) Math //
cosh(x)  Returns the hyperbolic cosine of x  Math //
count()  Logs the number of times that this particular call to //
count() has been called Console
createAttribute()  Creates an attribute node Document //
createComment()  Creates a Comment node with the specified text //
Document
createDocumentFragment() Creates an empty DocumentFragment node //
Document
createElement()  Creates an Element node Document //
createEvent()  Creates a new event Document, Event //
createTextNode() Creates a Text node Document //
```

```
ctrlKey  Returns whether the "CTRL" key was pressed when the mouse //
event was triggered MouseEvent, KeyboardEvent, TouchEvent
currentTarget  Returns the element whose event listeners triggered //
the event Event
data Returns the inserted characters InputEvent //
dataTransfer Returns an object containing the data being //
dragged/dropped, or inserted/deleted  DragEvent, InputEvent
debugger Stops the execution of JavaScript, and calls (if available) //
the debugging function  Statements
decodeURI()  Decodes a URI Global //
decodeURIComponent() Decodes a URI component Global //
defaultPrevented Returns whether or not the preventDefault() method //
was called for the event Event
defaultStatus  Sets or returns the default text in the statusbar of //
a window Window
defaultView  Returns the window object associated with a document, //
or null if none is available. Document
delete Deletes a property from an object Operators //
deltaX Returns the horizontal scroll amount of a mouse wheel //
(x-axis)  WheelEvent
deltaY Returns the vertical scroll amount of a mouse wheel (y-axis) //
WheelEvent
deltaZ Returns the scroll amount of a mouse wheel for the z-axis //
WheelEvent
deltaMode  Returns a number that represents the unit of measurements //
for delta values (pixels, lines or pages) WheelEvent
designMode Controls whether the entire document should be editable //
or not. Document
detail Returns a number that indicates how many times the mouse was //
clicked  UiEvent
do ... while Executes a block of statements and repeats the block //
while a condition is true  Statements
doctype  Returns the Document Type Declaration associated with the //
document  Document
document Returns the Document object for the window (See Document //
object)  Window
documentElement  Returns the Document Element of the document (the //
<html> element) Document
documentMode Returns the mode used by the browser to render the //
document Document
documentURI  Sets or returns the location of the document  Document //
domain Returns the domain name of the server that loaded the //
document  Document
```

```
domConfig  Obsolete. Returns the DOM configuration of the document //
Document
elapsedTime  Returns the number of seconds an animation has been //
running AnimationEvent
elapsedTime  Returns the number of seconds a transition has been //
running
embeds Returns a collection of all <embed> elements the document //
Document
encodeURI()  Encodes a URI Global //
encodeURIComponent() Encodes a URI component Global //
E  Returns Euler's number (approx. 2.718)  Math //
endsWith() Checks whether a string ends with specified //
string/characters String
entries()  Returns a key/value pair Array Iteration Object Array //
error()  Outputs an error message to the console Console //
escape() Deprecated in version 1.5. Use encodeURI() or //
encodeURIComponent() instead  Global
eval() Evaluates a string and executes it as if it was script code //
Global
eventPhase Returns which phase of the event flow is currently being //
evaluated  Event
every()  Checks if every element in an array pass a test Array //
exec() Tests for a match in a string. Returns the first match //
RegExp
execCommand()  Invokes the specified clipboard operation on the //
element currently having focus.  Document
exitFullscreen() Cancels an element in fullscreen mode Element //
exp(x) Returns the value of Ex Math //
expm1(x) Returns the value of Ex minus 1 Math //
export Export functions so they can be used for imports in external //
modules, and other scripts
extends  Extends a class (inherit) Classes //
dir  Sets or returns the value of the dir attribute of an element //
Element
fill() Fill the elements in an array with a static value Array //
filter() Creates a new array with every element in an array that //
pass a test Array
find() Returns the value of the first element in an array that pass //
a test Array
findIndex()  Returns the index of the first element in an array that //
pass a test Array
floor()  Returns x, rounded downwards to the nearest integer Math //
focus()  Gives focus to an element Element, Window //
```

```
for  Marks a block of statements to be executed as long as a //
condition is true Statements
for ... in Marks a block of statements to be executed for each //
element of an object (or array) Statements
forEach()  Calls a function for each array element Array //
forms  Returns a collection of all <form> elements in the document //
Document
forward()  Loads the next URL in the history list  History //
frameElement Returns the <iframe> element in which the current //
window is inserted  Window
frames Returns all <iframe> elements in the current window Window //
from() Creates an array from an object Array //
fromCharCode() Converts Unicode values to characters String //
fround() Returns the nearest (32-bit single precision) float //
representation of a number  Math
fullscreenElement  Returns the current element that is displayed in //
fullscreen mode  Document
fullscreenEnabled()  Returns a Boolean value indicating whether the //
document can be viewed in fullscreen mode  Document
function Declares a function Statements //
geolocation  Returns a Geolocation object that can be used to locate //
the user's position Navigator
getDate()  Returns the day of the month (from 1-31)  Date //
getDay() Returns the day of the week (from 0-6)  Date //
getAttribute() Returns the specified attribute value of an element //
node  Element
getAttributeNode() Returns the specified attribute node  Element //
getBoundingClientRect()  Returns the size of an element and its //
position relative to the viewport  Element
getComputedStyle() Gets the current computed CSS styles applied to //
an element  Window
getCurrentPosition() Returns the current position of the device //
Geolocation
getElementById() Returns the element that has the ID attribute with //
the specified value  Document
getElementsByClassName() Returns a NodeList containing all elements //
with the specified class name  Document, Element
getElementsByName()  Returns a NodeList containing all elements with //
a specified name  Document
getElementsByTagName() Returns a NodeList containing all elements //
with the specified tag name  Document, Element
getFullYear()  Returns the year  Date //
getHours() Returns the hour (from 0-23)  Date //
```

```
getItem()  Returns the value of the specified key name Storage //
getMilliseconds()  Returns the milliseconds (from 0-999) Date //
getMinutes() Returns the minutes (from 0-59) Date //
getModifierState() Returns an array containing target ranges that //
will be affected by the insertion/deletion MouseEvent
getMonth() Returns the month (from 0-11) Date //
getNamedItem() Returns a specified attribute node from a //
NamedNodeMap  Attribute
getSeconds() Returns the seconds (from 0-59) Date //
getSelection() Returns a Selection object representing the range of //
text selected by the user  Window
getTargetRanges()  Returns an array containing target ranges that //
will be affected by the insertion/deletion InputEvent
getTime()  Returns the number of milliseconds since midnight Jan 1 //
1970, and a specified date  Date
getTimezoneOffset()  Returns the time difference between UTC time //
and local time, in minutes Date
getUTCDate() Returns the day of the month, according to universal //
time (from 1-31) Date
getUTCDay()  Returns the day of the week, according to universal //
time (from 0-6) Date
getUTCFullYear() Returns the year, according to universal time Date //
getUTCHours()  Returns the hour, according to universal time (from //
0-23) Date
getUTCMilliseconds() Returns the milliseconds, according to //
universal time (from 0-999)  Date
getUTCMinutes()  Returns the minutes, according to universal time //
(from 0-59)  Date
getUTCMonth()  Returns the month, according to universal time (from //
0-11)  Date
getUTCSeconds()  Returns the seconds, according to universal time //
(from 0-59)  Date
getYear()  Deprecated. Use the getFullYear() method instead  Date //
global Checks whether the "g" modifier is set  RegExp //
go() Loads a specific URL from the history list  History //
group()  Creates a new inline group in the console. This indents //
following console messages by an additional level, until
console.groupEnd() is called Console
groupCollapsed() Creates a new inline group in the console. However, //
the new group is created collapsed. The user will need to use the
disclosure button to expand it  Console
groupEnd() Exits the current inline group in the console Console //
```

```
hasAttribute() Returns true if an element has the specified //
attribute, otherwise false Element
hasAttributes()  Returns true if an element has any attributes, //
otherwise false  Element
hasChildNodes()  Returns true if an element has any child nodes, //
otherwise false Element
hasFocus() Returns a Boolean value indicating whether the document //
has focus Document
hash Sets or returns the anchor part (#) of a URL  Location //
head Returns the <head> element of the document  Document //
height Returns the total height of the screen  Screen //
history  Returns the History object for the window (See History //
object)  Window
host Sets or returns the hostname and port number of a URL Location //
hostname Sets or returns the hostname of a URL Location //
href Sets or returns the entire URL  Location //
id Sets or returns the value of the id attribute of an element //
Element
if ... else ... else if  Marks a block of statements to be executed //
depending on a condition Statements
ignoreCase Checks whether the "i" modifier is set  RegExp //
images Returns a collection of all <img> elements in the document //
Document
implementation Returns the DOMImplementation object that handles //
this document Document
import Import functions exported from an external module, and //
another script
importNode() Imports a node from another document  Document //
in Returns true if the specified property is in the specified //
object, otherwise false  Operators
includes() Check if an array contains the specified element  Array, //
String
indexOf()  Search the array for an element and returns its position //
Array, String
Infinity A numeric value that represents positive/negative infinity //
Global
info() Outputs an informational message to the console Console //
innerHeight  Returns the height of the window's content area //
(viewport) including scrollbars Window
innerHTML  Sets or returns the content of an element Element //
innerText  Sets or returns the text content of a node and its //
descendants  Element
```

```
innerWidth Returns the width of a window's content area (viewport) //
including scrollbars  Window
inputEncoding  Returns the encoding, character set, used for the //
document  Document
inputType  Returns the type of the change (i.e "inserting" or //
"deleting")  InputEvent
insertAdjacentElement()  Inserts a HTML element at the specified //
position relative to the current element  Element
insertAdjacentHTML() Inserts a HTML formatted text at the specified //
position relative to the current element Element
insertAdjacentText() Inserts text into the specified position //
relative to the current element  Element
insertBefore() Inserts a new child node before a specified, //
existing, child node Element
instanceof Returns true if the specified object is an instance of //
the specified object Operators
isArray()  Checks whether an object is an array  Array //
isComposing  Returns whether the state of the event is composing or //
not  InputEvent, KeyboardEvent
isContentEditable  Returns true if the content of an element is //
editable, otherwise false  Element
isDefaultNamespace() Returns true if a specified namespaceURI is the //
default, otherwise false  Element
isEqualNode()  Checks if two elements are equal  Element //
isFinite() Determines whether a value is a finite, legal number //
Global, Number
isId Returns true if the attribute is of type Id, otherwise it //
returns false Attribute
isInteger()  Checks whether a value is an integer  Number //
isNaN()  Determines whether a value is an illegal number Global, //
Number
isSafeInteger()  Checks whether a value is a safe integer  Number //
isSameNode() Checks if two elements are the same node  Element //
isSupported()  Returns true if a specified feature is supported on //
the element Element
isTrusted  Returns whether or not an event is trusted  Event //
item() Returns the attribute node at a specified index in a //
NamedNodeMap Attribute, HTMLCollection
join() Joins all elements of an array into a string  Array //
key  Returns the key value of the key represented by the event //
KeyboardEvent, StorageEvent
key()  Returns the name of the nth key in the storage  Storage //
```

```
keyCode  Returns the Unicode character code of the key that //
triggered the onkeypress event, or the Unicode key code of the key that
triggered the onkeydown or onkeyup event KeyboardEvent
keys() Returns a Array Iteration Object, containing the keys of the //
original array Array
lang Sets or returns the value of the lang attribute of an element //
Element
language Returns the language of the browser Navigator //
lastChild  Returns the last child node of an element Element //
lastElementChild Returns the last child element of an element //
Element
lastIndex  Specifies the index at which to start the next match //
RegExp
lastIndexOf()  Search the array for an element, starting at the end, //
and returns its position  Array, String
lastModified Returns the date and time the document was last //
modified  Document
length Sets or returns the number of elements in an array  Array, //
Attribute, History, HTMLCollection, Window, Storage
lengthComputable Returns whether the length of the progress can be //
computable or not ProgressEvent
let  Declares a variable inside brackets {} scope  Statements //
links  Returns a collection of all <a> and <area> elements in the //
document that have a href attribute  Document
LN2  Returns the natural logarithm of 2 (approx. 0.693)  Math //
LN10 Returns the natural logarithm of 10 (approx. 2.302) Math //
loaded Returns how much work has been loaded ProgressEvent //
localeCompare()  Compares two strings in the current locale  String //
localStorage Allows to save key/value pairs in a web browser. Stores //
the data with no expiration date  Window
location Returns the location of a key on the keyboard or device //
KeyboardEvent
location Returns the Location object for the window (See Location //
object)  Window
log()  Returns the natural logarithm of a number Math, Console //
log10()  Returns the base-10 logarithm of a number Math //
log1p()  Returns the natural logarithm of 1 + a number Math //
log2() Returns the base-2 logarithm of a number  Math //
LOG2E  Returns the base-2 logarithm of E (approx. 1.442) Math //
LOG10E Returns the base-10 logarithm of E (approx. 0.434)  Math //
map()  Creates a new array with the result of calling a function for //
each array element  Array
```

```
match()  Searches a string for a match against a regular expression, //
and returns the matches String
matches()  Returns a Boolean value indicating whether an element is //
matched by a specific CSS selector or not  Element
matchMedia() Returns a MediaQueryList object representing the //
specified CSS media query string Window
max()  Returns the number with the highest value Math //
MAX_VALUE  Returns the largest number possible in JavaScript Number //
message  Sets or returns an error message (a string) Error //
metaKey  Returns whether the "META" key was pressed when an event //
was triggered  MouseEvent, KeyboardEvent, TouchEvent
min()  Returns the number with the lowest value  Math //
multiline  Checks whether the "m" modifier is set  RegExp //
MIN_VALUE  Returns the smallest number possible in JavaScript //
Number
moveBy() Moves a window relative to its current position Window //
moveTo() Moves a window to the specified position  Window //
MovementX  Returns the horizontal coordinate of the mouse pointer //
relative to the position of the last mousemove event MouseEvent
MovementY  Returns the vertical coordinate of the mouse pointer //
relative to the position of the last mousemove event MouseEvent
name Sets or returns an error name Error, Attribute, Window //
namedItem()  Returns the element with the specified ID, or name, in //
an HTMLCollection  HTMLCollection
namespaceURI Returns the namespace URI of an element Element //
NaN  "Not-a-Number" value  Global, Number //
navigator  Returns the Navigator object for the window (See //
Navigator object)  Window
NEGATIVE_INFINITY  Represents negative infinity (returned on //
overflow) Number
new  Creates an instance of a constructor //
newURL Returns the URL of the document, after the hash has been //
changed  HasChangeEvent
newValue Returns the new value of the changed storage item //
StorageEvent
nextSibling  Returns the next node at the same node tree level //
Element
nextElementSibling Returns the next element at the same node tree //
level  Element
nodeName Returns the name of a node  Element //
nodeType Returns the node type of a node Element //
nodeValue  Sets or returns the value of a node Element //
```

```
normalize()  Removes empty Text nodes, and joins adjacent nodes //
Document, Element
normalizeDocument()  Removes empty Text nodes, and joins adjacent //
nodes  Document
now()  Returns the number of milliseconds since midnight Jan 1, 1970 //
Date
Number() Converts an object's value to a number  Global //
offsetHeight Returns the height of an element, including padding, //
border and scrollbar Element
offsetWidth  Returns the width of an element, including padding, //
border and scrollbar  Element
offsetLeft Returns the horizontal offset position of an element //
Element
offsetParent Returns the offset container of an element  Element //
offsetTop  Returns the vertical offset position of an element //
Element
offsetX  Returns the horizontal coordinate of the mouse pointer //
relative to the position of the edge of the target element MouseEvent
offsetY  Returns the vertical coordinate of the mouse pointer //
relative to the position of the edge of the target element MouseEvent
oldURL Returns the URL of the document, before the hash was changed //
HasChangeEvent
oldValue Returns the old value of the changed storage item //
StorageEvent
onabort  The event occurs when the loading of a media is aborted //
UiEvent, Event
onafterprint The event occurs when a page has started printing, or //
if the print dialogue box has been closed Event
onanimationend The event occurs when a CSS animation has completed //
AnimationEvent
onanimationiteration The event occurs when a CSS animation is //
repeated AnimationEvent
onanimationstart The event occurs when a CSS animation has started //
AnimationEvent
onbeforeprint  The event occurs when a page is about to be printed //
Event
onbeforeunload The event occurs before the document is about to be //
unloaded  UiEvent, Event
onblur The event occurs when an element loses focus  FocusEvent //
oncanplay  The event occurs when the browser can start playing the //
media (when it has buffered enough to begin)  Event
oncanplaythrough The event occurs when the browser can play through //
the media without stopping for buffering Event
```

```
onchange The event occurs when the content of a form element, the //
selection, or the checked state have changed (for <input>, <select>,
and <textarea>) Event
onclick  The event occurs when the user clicks on an element //
MouseEvent
oncontextmenu  The event occurs when the user right-clicks on an //
element to open a context menu  MouseEvent
oncopy The event occurs when the user copies the content of an //
element ClipboardEvent
oncut  The event occurs when the user cuts the content of an element //
ClipboardEvent
ondblclick The event occurs when the user double-clicks on an //
element  MouseEvent
ondrag The event occurs when an element is being dragged DragEvent //
ondragend  The event occurs when the user has finished dragging an //
element DragEvent
ondragenter  The event occurs when the dragged element enters the //
drop target  DragEvent
ondragleave  The event occurs when the dragged element leaves the //
drop target  DragEvent
ondragover The event occurs when the dragged element is over the //
drop target DragEvent
ondragstart  The event occurs when the user starts to drag an //
element  DragEvent
ondrop The event occurs when the dragged element is dropped on the //
drop target DragEvent
ondurationchange The event occurs when the duration of the media is //
changed  Event
onemptied  The event occurs when something bad happens and the media //
(file is suddenly unavailable (like unexpectedly disconnects
onended  The event occurs when the media has reach the end (useful //
for messages like "thanks for listening") Event
onerror  The event occurs when an error occurs while loading an //
external file  ProgressEvent, UiEvent, Event
onfocus  The event occurs when an element gets focus FocusEvent //
onfocusin  The event occurs when an element is about to get focus //
FocusEvent
onfocusout The event occurs when an element is about to lose focus //
FocusEvent
onfullscreenchange The event occurs when an element is displayed in //
fullscreen mode  Event
onfullscreenerror  The event occurs when an element can not be //
displayed in fullscreen mode  Event
```

```
onhashchange The event occurs when there has been changes to the //
anchor part of a URL  HashChangeEvent
oninput  The event occurs when an element gets user input //
InputEvent, Event
oninvalid  The event occurs when an element is invalid Event //
onkeydown  The event occurs when the user is pressing a key //
KeyboardEvent
onkeypress The event occurs when the user presses a key //
KeyboardEvent
onkeyup  The event occurs when the user releases a key KeyboardEvent //
onLine Determines whether the browser is online  Navigator //
onload The event occurs when an object has loaded  UiEvent, Event //
onloadeddata The event occurs when media data is loaded  Event //
onloadedmetadata The event occurs when meta data (like dimensions //
and duration) are loaded Event
onloadstart  The event occurs when the browser starts looking for //
the specified media  ProgressEvent
onmessage  The event occurs when a message is received through the //
event source  Event
onmousedown  The event occurs when the user presses a mouse button //
over an element MouseEvent
onmouseenter The event occurs when the pointer is moved onto an //
element  MouseEvent
onmouseleave The event occurs when the pointer is moved out of an //
element  MouseEvent
onmousemove  The event occurs when the pointer is moving while it is //
over an element MouseEvent
onmouseover  The event occurs when the pointer is moved onto an //
element, or onto one of its children MouseEvent
onmouseout The event occurs when a user moves the mouse pointer out //
of an element, or out of one of its children MouseEvent
onmouseup  The event occurs when a user releases a mouse button over //
an element  MouseEvent
onmousewheel Deprecated. Use the wheel event instead WheelEvent //
onoffline  The event occurs when the browser starts to work offline //
Event
ononline The event occurs when the browser starts to work online //
Event
onopen The event occurs when a connection with the event source is //
opened  Event
onpagehide The event occurs when the user navigates away from a //
webpage  PageTransitionEvent
```

```
onpageshow The event occurs when the user navigates to a webpage //
PageTransitionEvent
onpaste  The event occurs when the user pastes some content in an //
element  ClipboardEvent
onpause  The event occurs when the media is paused either by the //
user or programmatically  Event
onplay The event occurs when the media has been started or is no //
longer paused Event
onplaying  The event occurs when the media is playing after having //
been paused or stopped for buffering  Event
onpopstate The event occurs when the window's history changes //
PopStateEvent
onprogress The event occurs when the browser is in the process of //
getting the media data (downloading the media) Event
onratechange The event occurs when the playing speed of the media is //
changed Event
onresize The event occurs when the document view is resized //
UiEvent, Event
onreset  The event occurs when a form is reset Event //
onscroll The event occurs when an element's scrollbar is being //
scrolled  UiEvent, Event
onsearch The event occurs when the user writes something in a search //
field (for <input="search">)  Event
onseeked The event occurs when the user is finished moving/skipping //
to a new position in the media Event
onseeking  The event occurs when the user starts moving/skipping to //
a new position in the media  Event
onselect The event occurs after the user selects some text (for //
<input> and <textarea>)  UiEvent, Event
onshow The event occurs when a <menu> element is shown as a context //
menu Event
onstalled  The event occurs when the browser is trying to get media //
data, but data is not available  Event
onstorage  The event occurs when a Web Storage area is updated //
StorageEvent
onsubmit The event occurs when a form is submitted Event //
onsuspend  The event occurs when the browser is intentionally not //
getting media data Event
ontimeupdate The event occurs when the playing position has changed //
(like when the user fast forwards to a different point in the media)
Event
ontoggle The event occurs when the user opens or closes the //
<details> element  Event
```

```
ontouchcancel  The event occurs when the touch is interrupted //
TouchEvent
ontouchend The event occurs when a finger is removed from a touch //
screen TouchEvent
ontouchmove  The event occurs when a finger is dragged across the //
screen TouchEvent
ontouchstart The event occurs when a finger is placed on a touch //
screen  TouchEvent
ontransitionend  The event occurs when a CSS transition has //
completed  TransitionEvent
onunload The event occurs once a page has unloaded (for <body>) //
UiEvent, Event
onvolumechange The event occurs when the volume of the media has //
changed (includes setting the volume to "mute") Event
onwaiting  The event occurs when the media has paused but is //
expected to resume (like when the media pauses to buffer more data)
Event
onwheel  The event occurs when the mouse wheel rolls up or down over //
an element  WheelEvent
open() Opens an HTML output stream to collect output from //
document.write() Document, Window
opener Returns a reference to the window that created the window //
Window
origin Returns the protocol, hostname and port number of a URL //
Location
outerHeight  Returns the height of the browser window, including //
toolbars/scrollbars Window
outerHTML  Sets or returns the outer content of an element Element //
outerText  Sets or returns the text outer content of a node and its //
descendants  Element
outerWidth Returns the width of the browser window, including //
toolbars/scrollbars  Window
ownerDocument  Returns the root element (document object) for an //
element Element
pageX  Returns the horizontal coordinate of the mouse pointer, //
relative to the document, when the mouse event was triggered
MouseEvent
pageXOffset  Returns the pixels the current document has been //
scrolled (horizontally) from the upper left corner of the window Window
pageY  Returns the vertical coordinate of the mouse pointer, //
relative to the document, when the mouse event was triggered
MouseEvent
```

```
pageYOffset  Returns the pixels the current document has been //
scrolled (vertically) from the upper left corner of the window Window
parent Returns the parent window of the current window Window //
parentNode Returns the parent node of an element Element //
parentElement  Returns the parent element node of an element Element //
parse()  Parses a date string and returns the number of milliseconds //
since January 1, 1970 Date, JSON
parseFloat() Parses a string and returns a floating point number //
Global
parseInt() Parses a string and returns an integer  Global //
pathname Sets or returns the path name of a URL  Location //
persisted  Returns whether the webpage was cached by the browser //
PageTransitionEvent
PI Returns PI (approx. 3.14) Math //
pixelDepth Returns the color resolution (in bits per pixel) of the //
screen  Screen
platform Returns for which platform the browser is compiled //
Navigator
pop()  Removes the last element of an array, and returns that //
element  Array
port Sets or returns the port number of a URL  Location //
position Returns the position of the concerned device at a given //
time  Geolocation
positionError  Returns the reason of an error occurring when using //
the geolocating device  Geolocation
positionOptions  Describes an object containing option properties to //
pass as a parameter of Geolocation.getCurrentPosition() and
Geolocation.watchPosition() Geolocation
POSITIVE_INFINITY  Represents infinity (returned on overflow) //
Number
preventDefault() Cancels the event if it is cancelable, meaning that //
the default action that belongs to the event will not occur Event
print()  Prints the content of the current window  Window //
product  Returns the engine name of the browser  Navigator //
propertyName Returns the name of the CSS property associated with //
the animation or transition  AnimationEvent, TransitionEvent
protocol Sets or returns the protocol of a URL Location //
prototype  Allows you to add properties and methods to an Array //
object Array, Boolean, Date
pseudoElement  Returns the name of the pseudo-element of the //
animation or transition AnimationEvent, TransitionEvent
push() Adds new elements to the end of an array, and returns the new //
length  Array
```

```
querySelector()  Returns the first element that matches a specified //
CSS selector(s) in the document  Document, Element
querySelectorAll() Returns a static NodeList containing all elements //
that matches a specified CSS selector(s) in the document  Document,
Element
random() Returns a random number between 0 and 1 Math //
readyState Returns the (loading) status of the document  Document //
reduce() Reduce the values of an array to a single value (going //
left-to-right) Array
reduceRight()  Reduce the values of an array to a single value //
(going right-to-left) Array
referrer Returns the URL of the document that loaded the current //
document  Document
region   MouseEvent //
reload() Reloads the current document  Location //
remove() Removes the element from the DOM  Element //
removeAttribute()  Removes a specified attribute from an element //
Element
removeAttributeNode()  Removes a specified attribute node, and //
returns the removed node  Element
removeChild()  Removes a child node from an element  Element //
removeEventListener()  Removes an event handler that has been //
attached with the addEventListener() method  Element
removeItem() Removes that key from the storage Storage //
repeat Returns whether a key is being hold down repeatedly, or not //
KeyboardEvent
repeat() Returns a new string with a specified number of copies of //
an existing string  String
replace()  Searches a string for a specified value, or a regular //
expression, and returns a new string where the specified values are
replaced  String, Location
replaceChild() Replaces a child node in an element Element //
requestAnimationFrame()  Requests the browser to call a function to //
update an animation before the next repaint  Window
requestFullscreen()  Shows an element in fullscreen mode Element //
resizeBy() Resizes the window by the specified pixels  Window //
resizeTo() Resizes the window to the specified width and height //
Window
return Stops the execution of a function and returns a value from //
that function  Statements
reverse()  Reverses the order of the elements in an array  Array //
round()  Rounds x to the nearest integer Math //
pow()  Returns the value of x to the power of y  Math //
```

```
previousSibling  Returns the previous node at the same node tree //
level Element
previousElementSibling Returns the previous element at the same node //
tree level  Element
prompt() Displays a dialog box that prompts the visitor for input //
Window
prototype  Allows you to add properties and methods to an object //
Number
relatedTarget  Returns the element related to the element that //
triggered the mouse event MouseEvent, FocusEvent
removeEventListener()  Removes an event handler from the document //
(that has been attached with the addEventListener() method)  Document
removeNamedItem()  Removes a specified attribute node  Attribute //
renameNode() Renames the specified node  Document //
screen Returns the Screen object for the window (See Screen object) //
Window
screenLeft Returns the horizontal coordinate of the window relative //
to the screen  Window
screenTop  Returns the vertical coordinate of the window relative to //
the screen  Window
screenX  Returns the horizontal coordinate of the window/mouse //
pointer relative to the screen  Window, MouseEvent
screenY  Returns the vertical coordinate of the window/mouse pointer //
relative to the screen  Window, MouseEvent
scripts  Returns a collection of <script> elements in the document //
Document
scroll() Deprecated. This method has been replaced by the scrollTo() //
method. Window
scrollBy() Scrolls the document by the specified number of pixels //
Window
scrollHeight Returns the entire height of an element, including //
padding  Element
scrollIntoView() Scrolls the specified element into the visible area //
of the browser window Element
scrollLeft Sets or returns the number of pixels an element's content //
is scrolled horizontally  Element
scrollTo() Scrolls the document to the specified coordinates Window //
scrollTop  Sets or returns the number of pixels an element's content //
is scrolled vertically  Element
scrollWidth  Returns the entire width of an element, including //
padding Element
scrollX  An alias of pageXOffset Window //
scrollY  An alias of pageYOffset Window //
```

```
search Sets or returns the querystring part of a URL Location //
search() Searches a string for a specified value, or regular //
expression, and returns the position of the match String
self Returns the current window  Window //
sessionStorage Allows to save key/value pairs in a web browser. //
Stores the data for one session  Window
setAttribute() Sets or changes the specified attribute, to the //
specified value Element
setAttributeNode() Sets or changes the specified attribute node //
Element
setDate()  Sets the day of the month of a date object  Date //
setFullYear()  Sets the year of a date object  Date //
setHours() Sets the hour of a date object  Date //
setInterval()  Calls a function or evaluates an expression at //
specified intervals (in milliseconds)  Window
setItem()  Adds that key to the storage, or update that key's value //
if it already exists Storage
setMilliseconds()  Sets the milliseconds of a date object  Date //
setMinutes() Set the minutes of a date object  Date //
setMonth() Sets the month of a date object Date //
setNamedItem() Sets the specified attribute node (by name) Attribute //
setSeconds() Sets the seconds of a date object Date //
setTime()  Sets a date to a specified number of milliseconds //
after/before January 1, 1970  Date
setTimeout() Calls a function or evaluates an expression after a //
specified number of milliseconds  Window
setUTCDate() Sets the day of the month of a date object, according //
to universal time Date
setUTCFullYear() Sets the year of a date object, according to //
universal time Date
setUTCHours()  Sets the hour of a date object, according to //
universal time Date
setUTCMilliseconds() Sets the milliseconds of a date object, //
according to universal time Date
setUTCMinutes()  Set the minutes of a date object, according to //
universal time Date
setUTCMonth()  Sets the month of a date object, according to //
universal time  Date
setUTCSeconds()  Set the seconds of a date object, according to //
universal time Date
setYear()  Deprecated. Use the setFullYear() method instead  Date //
shift()  Removes the first element of an array, and returns that //
element Array
```

```
shiftKey Returns whether the "SHIFT" key was pressed when an event //
was triggered MouseEvent, KeyboardEvent, TouchEvent
sign(x)  Returns the sign of a number (checks whether it is //
positive, negative or zero)  Math
sin()  Returns the sine of x (x is in radians) Math //
sinh() Returns the hyperbolic sine of x  Math //
slice()  Selects a part of an array, and returns the new array //
Array, String
some() Checks if any of the elements in an array pass a test Array //
sort() Sorts the elements of an array  Array //
source Returns the text of the RegExp pattern  RegExp //
specified  Returns true if the attribute has been specified, //
otherwise it returns false  Attribute
splice() Adds/Removes elements from an array Array //
split()  Splits a string into an array of substrings String //
static Defines a static method for a class Classes //
startsWith() Checks whether a string begins with specified //
characters  String
state  Returns an object containing a copy of the history entries //
PopStateEvent
String() Converts an object's value to a string  Global //
stop() Stops the window from loading Window //
stopImmediatePropagation() Prevents other listeners of the same //
event from being called  Event
stopPropagation()  Prevents further propagation of an event during //
event flow  Event
stringify()  Convert a JavaScript object to a JSON string  JSON //
sqrt() Returns the square root of x  Math //
SQRT1_2  Returns the square root of 1/2 (approx. 0.707)  Math //
SQRT2  Returns the square root of 2 (approx. 1.414)  Math //
status Sets or returns the text in the statusbar of a window Window //
strictErrorChecking  Sets or returns whether error-checking is //
enforced or not Document
storageArea  Returns an object representing the affected storage //
object  StorageEvent
style  Sets or returns the value of the style attribute of an //
element  Element
substr() Extracts the characters from a string, beginning at a //
specified start position, and through the specified number of character
String
substring()  Extracts the characters from a string, between two //
specified indices  String
super  Refers to the parent class  Classes //
```

```
switch Marks a block of statements to be executed depending on //
different cases Statements
table()  Displays tabular data as a table  Console //
tabIndex Sets or returns the value of the tabindex attribute of an //
element Element
tagName  Returns the tag name of an element  Element //
tan()  Returns the tangent of an angle Math //
tanh() Returns the hyperbolic tangent of a number  Math //
target Returns the element that triggered the event  Event //
targetTouches  Returns a list of all the touch objects that are in //
contact with the surface and where the touchstart event occured on the
same target element as the current target element  TouchEvent
test() Tests for a match in a string. Returns true or false  RegExp //
textContent  Sets or returns the textual content of a node and its //
descendants Element
this Refers to the object it belongs to //
throw  Throws (generates) an error Statements //
time() Starts a timer (can track how long an operation takes) //
Console
timeEnd()  Stops a timer that was previously started by //
console.time() Console
timeStamp  Returns the time (in milliseconds relative to the epoch) //
at which the event was created Event
title  Sets or returns the title of the document Document, Element //
toDateString() Converts the date portion of a Date object into a //
readable string Date
toGMTString()  Deprecated. Use the toUTCString() method instead //
Date
toExponential()  Converts a number into an exponential notation //
Number
toFixed(x) Formats a number with x numbers of digits after the //
decimal point Number
toJSON() Returns the date as a string, formatted as a JSON date //
Date
toISOString()  Returns the date as a string, using the ISO standard //
Date
toLocaleDateString() Returns the date portion of a Date object as a //
string, using locale conventions Date
toLocaleLowerCase()  Converts a string to lowercase letters, //
according to the host's locale  String
toLocaleString() Converts a Date object to a string, using locale //
conventions  Date
```

```
toLocaleTimeString() Returns the time portion of a Date object as a //
string, using locale conventions Date
toLocaleUpperCase()  Converts a string to uppercase letters, //
according to the host's locale  String
toLowerCase()  Converts a string to lowercase letters  String //
top  Returns the topmost browser window  Window //
toPrecision(x) Formats a number to x length  Number //
toString() Converts an array to a string, and returns the result //
Array, Boolean, Date, Number, RegExp, String, Element
total  Returns the total amount of work that will be loaded //
ProgressEvent
toTimeString() Converts the time portion of a Date object to a //
string  Date
touches  Returns a list of all the touch objects that are currently //
in contact with the surface  TouchEvent
toUpperCase()  Converts a string to uppercase letters  String //
toUTCString()  Converts a Date object to a string, according to //
universal time Date
trace()  Outputs a stack trace to the console  Console //
transitionend  The event occurs when a CSS transition has completed //
TransitionEvent
trim() Removes whitespace from both ends of a string String //
trunc()  Returns the integer part of a number (x)  Math //
try ... catch ... finally  Marks the block of statements to be //
executed when an error occurs in a try block, and implements error
handling Statements
type Returns the name of the event Event //
typeof Returns the type of a variable, object, function or //
expression  Operators
undefined  Indicates that a variable has not been assigned a value //
Global
unescape() Deprecated in version 1.5. Use decodeURI() or //
decodeURIComponent() instead  Global
unshift()  Adds new elements to the beginning of an array, and //
returns the new length  Array
url  Returns the URL of the changed item's document  StorageEvent //
URL  Returns the full URL of the HTML document Document //
userAgent  Returns the user-agent header sent by the browser to the //
server Navigator
UTC()  Returns the number of milliseconds in a date since midnight //
of January 1, 1970, according to UTC time Date
value  Sets or returns the value of the attribute  Attribute //
```

```
valueOf()  Returns the primitive value of an array Array, Boolean, //
Date, Number, String
var  Declares a variable Statements //
warn() Outputs a warning message to the console  Console //
watchPosition()  Returns a watch ID value that then can be used to //
unregister the handler by passing it to the Geolocation.clearWatch()
method Geolocation
which  Returns which mouse button was pressed when the mouse event //
was triggered MouseEvent, KeyboardEvent
width  Returns the total width of the screen Screen //
view Returns a reference to the Window object where the event //
occurred UiEvent
void Evaluates an expression and returns undefined Operators //
while  Marks a block of statements to be executed while a condition //
is true  Statements
write()  Writes HTML expressions or JavaScript code to a document //
Document
writeln()  Same as write(), but adds a newline character after each //
statement  Document
yield  Pauses and resumes a generator function //

//**************************END*********************** //
```